Implementación de Funciones de Procesamiento y Manipulación de Espectros de Resonancia Magnética Nuclear dentro del Marco del Proyecto: "*nemo2: A tool for assignment and prediction of NMR spectra"*

Trabajo de grado

JHONATAN NOGUERA ROMERO
1329700
jhonatan.noguera@correounivalle.edu.co

Andres Mauricio Castillo
Doctor
andres.m.castillo@correounivalle.edu.co

Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación
Programa Académico de Ingeniería de Sistemas
Cali, Mayo 22 de 2018

Tabla de contenido

Lista de tablas

Lista de ilustraciónes

1. Introducción

1.1 Resumen

El presente documento expone un proyecto de trabajo de grado, el cual consiste en exponer los resultados obtenidos en la actualización de una aplicación existente para el manejo de espectros de resonancia magnética nuclear(RMN). Los objetivos de este proyecto son la actualización, la implementación y el desarrollo de funcionalidades para esta herramienta, que fue desarrollada usando applets en Java 1.1 hace cerca de 10 años y cuya interfaz gráfica ha quedado prácticamente obsoleta pues ya no es compatible con ninguna de las tecnologías dominantes en el mercado.

Basándose principalmente en el uso de JavaScript, Java 8 y JavaFX, se realiza una renovación de tecnologías que mejora en gran medida la interfaz gráfica de la aplicación sin incurrir en cambios significativos de la interacción humano-computador comparado con la versión anterior y la inclusión de nuevas tecnologías en el desarrollo para realizar las funcionalidades planteadas en este documento se encargaran de satisfacer nuevos requerimientos funcionales por parte de los usuarios.

1.2 Planteamiento y Formulación del Problema

La Ciencia es el conocimiento organizado y sistematizado del saber humano. Las diferentes ciencias, evolucionan gracias a los esfuerzos hacia la apropiación de la realidad del ser humano. La Química es una ciencia activa y en constante evolución de vital importancia para nuestro planeta, tanto su naturaleza como en la sociedad. A pesar de que sus orígenes sean antiguos, es en todo sentido una ciencia moderna. La Química es el estudio de la materia y los cambios que ocurren en ella (Chang, 2013). Es frecuentemente considerada una ciencia central debido a que sus conocimientos básicos son indispensables para el desarrollo de varias disciplinas como la biología, física, geología entre otras. Entre los diversos temas de la Química, existe una técnica que se basa en el estudio de los núcleos magnéticos, la cual es conocida como Resonancia Magnética Nuclear cuyos experimentos resultan en la generación de un espectro el cual permite encontrar información importante de las moléculas de la muestra en estudio. Para esto, existen diferentes aplicaciones capaces de analizar los espectros y facilitar el análisis de la información. Entre estas funciones, se distinguen algunos como la predicción de

desplazamientos químicos, simulación de las moléculas, asignación automática, detección de picos entre otros, permiten un mayor entendimiento de las muestras analizadas.

Existen diferentes opciones para trabajar con los espectros de resonancia magnética nuclear y obtener información a partir de ellos. Una alternativa, es la aplicación Mylims ideada y desarrollada en el año 2008 por, Luc Patiny, Damiano Banfi y Marco Engeler desde Suiza y Julien Wist y Andrés M. Castillo en Colombia. Mylims es una aplicación de carácter libre, que permite manipular espectros, datos, interpretar resultados y compararlos con espectros obtenidos por otros usuarios. Sin embargo, es necesario realizar una actualización tecnológica en una nueva versión del proyecto Nemo (Hook4, 2017), incluído en Mylims. Esto es debido a la falta de soporte ofrecido para la plataforma de desarrollo original y la poca compatibilidad que tiene con las tecnologías actuales. Además, la aplicación original usa applets lo cual es una tecnología declarada obsoleta (Oracle, 2016). Para esto, se ha decidido realizarlo bajo la plataforma de desarrollo Java 8 con la cual se migrarán las funcionalidades existentes en Nemo, tales como predicción de desplazamientos químicos, constantes de acople e integrales, detección de picos, simulación del espectro del sistema y asignación automática de frecuencias de un espectro con sus átomos correspondientes y eventualmente desarrollar una forma de permitir al programa incluir la opción de realizar elucidación en moléculas pequeñas.

1.3 Responsabilidad y similitud frente a otros trabajos

Debido a la naturaleza de este software, existe un segundo documento realizado por un integrante del equipo. La diferencia principal entre estos documentos, radica en que la finalidad de este se centra en la definición, mejora e implementación de funciones existentes en la versión previa de Nemo2. El otro trabajo se enfoca principalmente en el desarrollo de la interfaz, su interacción con los usuarios y la visualización realizada por las funciones a implementar.

Debido a que es el equipo de desarrollo fue de 4 personas y eventualmente perdió un integrante, existen similitudes entre los documentos, ya que el trabajo realizado por uno de los integrantes era comunicado en las reuniones de retrospectiva y en las revisiones.

Las características de scrum usadas, herramientas como el lenguaje, control de versiones y software utilizado para la creación de historias de usuario y backlog, fueron

definidas antes de iniciar el desarrollo en el proyecto y acordadas por los 4 integrantes, debido a esto, existe una gran similitud en estas secciones del documento.

1.3 Justificación del Problema

1.3.1 Justificación Académica

La elaboración de este trabajo permite aplicar los conocimientos y habilidades de diseño y desarrollo de software bajo metodologías ágiles, mientras nos incentiva a adquirir conocimientos de otras áreas del conocimiento.

1.3.2 Justificación Económica

Como Mylims, existen distintas herramientas para la manipulación de espectros de resonancia magnética nuclear. Sin embargo, la mayoría de estos programas requieren el pago de una licencia para su uso. Mylims ofrece las funcionalidades más comunes para la manipulación de espectros de resonancia magnética de forma gratuita y libre.

1.3.3 Justificación Social

Para la comunidad estudiantil, muchos encuentran difícil utilizar recursos para adquirir un software el cual no necesariamente se usará de manera intensiva. Si bien una institución universitaria puede adquirir un software pago, el estudiante puede ver su trabajo limitado desde casa. Mylims es una de las alternativas al momento de no tener la facilidad de adquirir una membresía para las herramientas comerciales. Mylims permite a los estudiantes cargar moléculas para estudiar su espectro de resonancia magnética y realizar análisis sobre estos. Además, el software Mylims, no solo es utilizado en la Universidad del Valle, este también es usado por la empresa Actelion de Suiza y la Ecole Polytecnique Fédérale de Laussane en Suiza. La actualización de esta aplicación, permitiría a estas tres instituciones el uso de esta nueva versión.

1.4 Objetivos

1.4.1 Objetivo General

Implementar las funciones existentes de procesamiento de espectros de RMN disponibles en la versión anterior de Mylims a una nueva versión de la herramienta que usará la tecnología JavaFx.

1.4.2 Objetivos Específicos

- Implementar en java 8, una nueva versión del algoritmo de detección de picos(peak-picking).
- Implementar en java 8, una nueva versión del algoritmo de asignación automática.
- Implementar en java 8, una nueva versión del predictor de desplazamientos químicos para 1H y 13C.
- Implementar las pruebas de unidad(JUnit) para cada una de las funciones descritas anteriormente.
- Diseñar e implementar una interfaz(API) que permita el uso de las funciones descritas anteriormente desde una interfaz gráfica de usuario y desde un procesador de datos en lotes(scripting) junto con las funciones que ya existen.

1.4.3 Resultados del proyecto

Objetivos Específicos	Resultados
Implementar en java el algoritmo de detección de picos (peak-picking).	Código fuente de la función e informe de documentación.
Implementar en java una nueva versión del algoritmo de asignación automática.	Código fuente de la función e informe de documentación.
Implementar en java un predictor de desplazamientos químicos para 1H y 13C basado en el algoritmo.	Código fuente de la función e informe de documentación.
Implementar las pruebas de unidad (JUnit) para cada una de las funciones descritas anteriormente.	Conjunto de pruebas unitarias por cada una de las funciones anteriores.
Diseñar e implementar una interfaz (API) que permita el uso de las funciones descritas anteriormente.	Código fuente y ubicación donde se encuentra disponible la API para su uso.

Tabla 1 Resultados del proyecto

1.5 Alcances del proyecto

La propuesta especificada en este proyecto, se limita a la actualización del programa Nemo2 y la implementación de detección de picos, asignación automática y predicción de desplazamientos químicos, así mismo, como sus pruebas de unidad y una API que permita su utilización.

2. Marco referencial

2.1 Antecedentes o Estado del Arte

ACD/Spectrus:

Advanced Chemistry Development, Inc. (ACD/Labs) es la empresa que desarrolló esta herramienta, cuenta con un amplio portafolio de productos y servicios entre los cuales se encuentra ACD/Spectrus. Este permite “los datos procesados y sin procesar de diferentes instrumentos sean combinados en un solo ambiente uniforme para su procesamiento, interpretación, reportes, almacenamiento y reutilización”.

ACD/Labs hace uso de la unificación de datos químicos, estructurales y analíticos, el uso de una visualización dinámica y algoritmos avanzados para presentar información de manera precisa para que los investigadores puedan acceder, interpretar y usar los resultados de la mejor manera (Acd/Labs, 1996-2017).

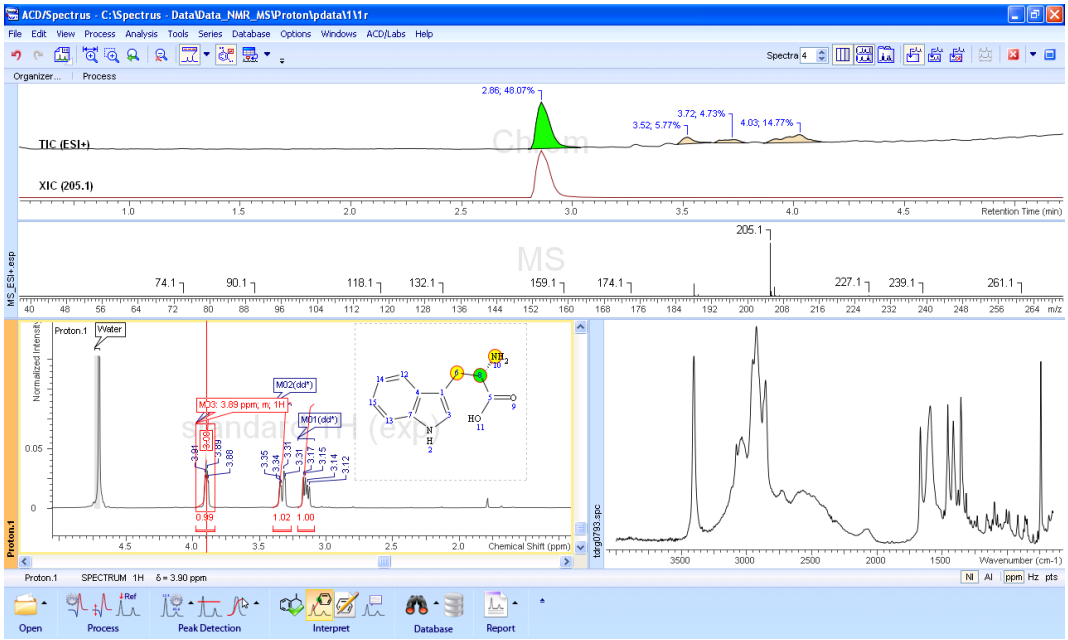

Ilustración 1 Interfaz de ACD/Spectrus Platform (Tomado de: http://www.acdlabs.com/images/products/uli/uli_ss1.png)

Mestrenova:

Desarrollado por MestreLab. MestreNova es un software que permite procesar datos químicos analíticos. Permite analizar una variedad de módulos químicos avanzados como análisis de mezclas, monitoréo de reacción, predicción de desplazamientos químicos entre otros. Adicionalmente, MNova permite compartir la interfaz principal para tener un acceso más rápido a los datos necesarios.

MNova usa algoritmos avanzados para proveer resultados de alta calidad, que al ser combinados con un software que permite realizar distintos estudios al tiempo, provee

una alta productividad al tener los datos necesarios a la mano. Posee la posibilidad de preparar un flujo de trabajo automatizado o la manipulación de los datos manualmente.

MNova posee una cantidad de plugins instalables al software para actividades en específicas que abarcan un rango básico ha avanzado los cuales pueden ser adquiridos por separado para satisfacer las necesidades que el cliente necesita (MestreLab Research, 2016).

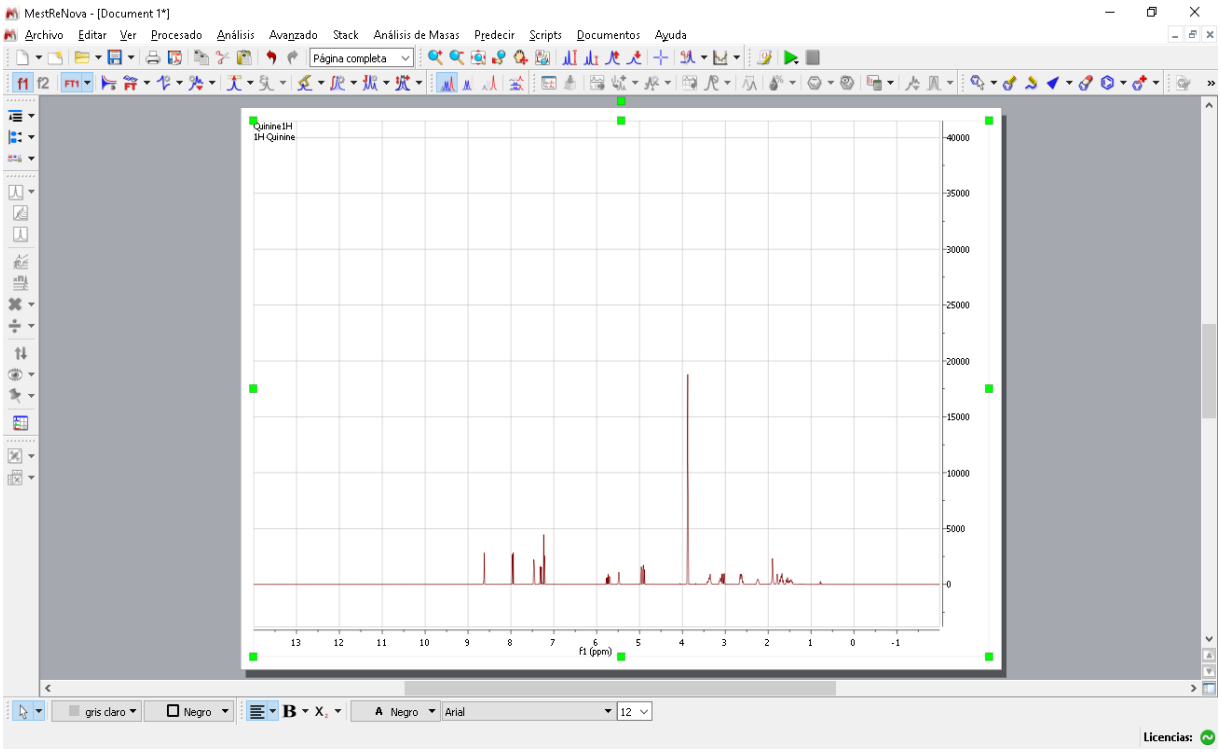

Ilustración 2Interfaz de MestreNova (Tomado de: Aplicación de escritorio MestreNova)

NMRViewJ:

Desarrollado por One Moon Scientifics, permite la visualización y el análisis de datos de NMR en una ventana multi canvas, logrando así mostrar varios espectros en una misma ventana de la aplicación. Esta también permite el uso de diferentes funcionalidades como la manipulación del diseño de los espectros como el color, contorno, etc, la generación de gráficos, identificación de picos, desplazamientos químicos, entre otras herramientas de análisis. Esta herramienta fue desarrollada en el lenguaje de programación Java ya que para la compañía seria mas facil que funcionara en los diferentes sistemas operativos (One Moon Scientific, 2016).

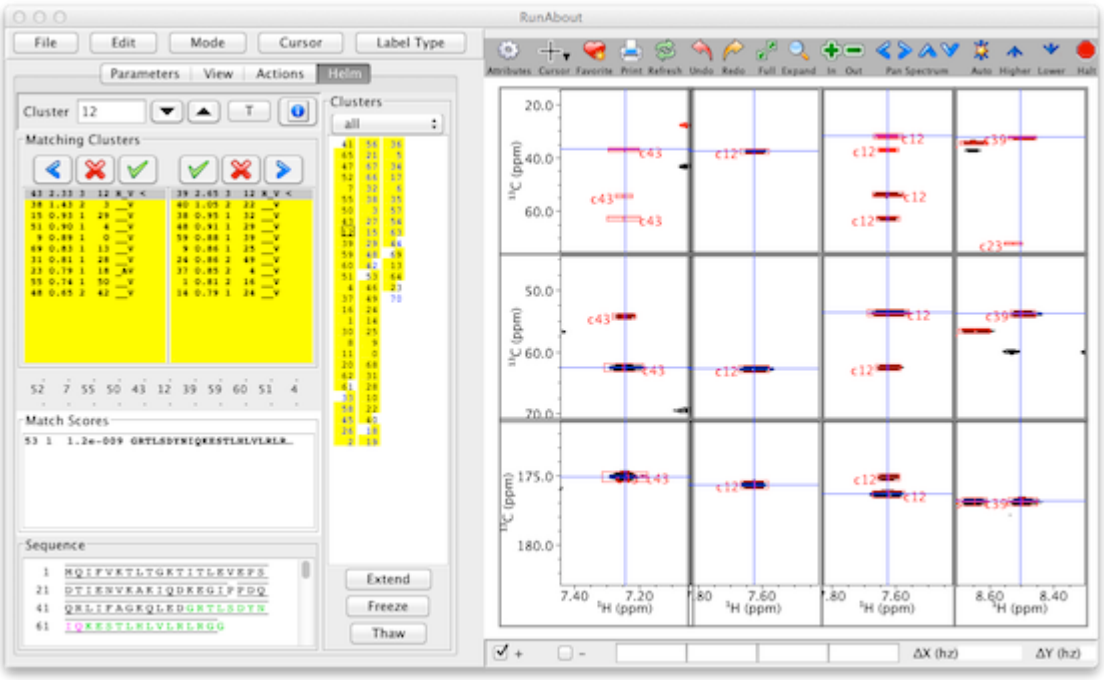

Ilustración 3 Interfaz de NRMView (Tomado de: http://www.onemoonscientific.com/images/omsimages/Banner_NVJ_560x345.png)

PERCH:

Desarrollado por PERCH Solutions, es una poderosa herramienta que provee información cualitativa y estructural para la caracterización de compuestos y mezclas. Se proveen los desplazamientos químicos y constantes de acople para espectro 1D - 1H con información estructural fácilmente comprensible. Entre otras de sus funciones, podemos encontrar un editor de estructura 4D, modelado molecular, predicción avanzada de NMR de protones, análisis de espectros y una interfaz amigable (PERCH solutions Ltd, s.f.).

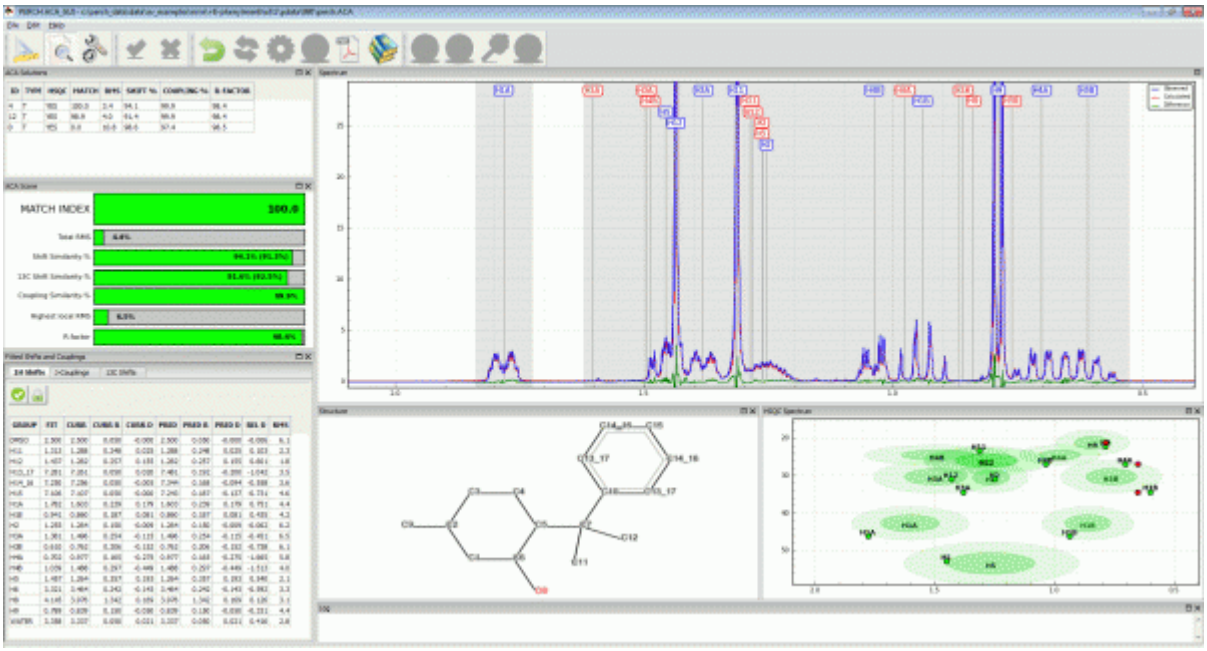

Ilustración 4 Interfaz de PERCHNMR (Tomado de: Aplicación de escritorio PERCHNMR)

2.2 Marco Teórico

Metodologías ágiles

El término ágil, nació en febrero de 2001 en una conferencia realizada en Utah, EEUU. Este tipo de metodologías, surgieron como alternativa a los métodos de desarrollo de software tradicionales. Basándose en que el proceso de desarrollo de software debe ser rápido, de calidad y pueda enfrentarse a los cambios que puedan surgir durante el proceso. Una metodología ágil, reduce el riesgo de fracaso debido a subestimación de costos, tiempo y funcionalidades del proyecto a desarrollar. Los valores consignados para cada una de estas metodologías son los siguientes:

- Formar el equipo de desarrollo y permitirles que ellos configuren su entorno en base a sus necesidades.
- No producir documentos a menos que sean necesarios para una toma de decisiones inmediata. De hacerse, deben ser cortos.
- La interacción constante entre cliente y el equipo de desarrollo asegurará el éxito del proyecto.
- Debe existir una planificación flexible y abierta, capaz de resistir cambios que se puedan generar en el transcurso del proceso.

Una metodología ágil, se subdivide en varios subproyectos que se tratan de manera independiente y desarrolla un subconjunto de características en un lapso de tiempo corto. A diferencia de una metodología tradicional, las metodologías ágiles son procesos altamente colaborativos que desean la existencia de cambios en los requerimientos para cumplir con las expectativas del cliente a su gusto. Se planean entregas constantes al cliente lo cual mantiene una comunicación entre este y el equipo de desarrollo conocida como retroalimentación (Cadavid, Martínez, Vélez, 2013).

En la tabla 1. Podemos encontrar una comparación de los puntos de contraste más importantes entre las metodologías ágiles y las tradicionales.

Metodologías ágiles	**Metodologías tradicionales**
Proceso Flexible	Proceso rígido
Basado en personas	Basado en procesos
El cliente es parte del equipo	Interacción mínima con el cliente
Poca documentación	Extensa documentación
Entregas constantes	Una sola entrega final
Reglas impuestas por el equipo	Reglas impuestas externamente
Proceso poco controlado	Proceso con altas reglas y normatividades
No existe un contrato tradicional	Contrato prefijado
Grupos pequeños	Grandes grupos
Pocos roles	Gran cantidad de roles
Menos énfasis en la arquitectura	La arquitectura es esencial

Tabla 2 Comparación entre metodología tradicional y ágil

Scrum

Scrum es un método de desarrollo ágil, simple y efectivo para la colaboración de un equipo en productos complejos. Este método se basa en unos valores, roles, eventos y artefactos que se usan para mantener un orden y entregar productos del mayor valor posible. Scrum es una de las metodologías más comunes en la industria del desarrollo de software debido

a la flexibilidad que presenta, la alta interacción entre el equipo de desarrollo y la inclusión del cliente en este proceso.

Es importante definir las características que hacen de esta metodología apreciada por los desarrolladores, entre estas encontramos los valores de Scrum:

- Coraje: Hacer lo correcto y trabajar en problemas difíciles
- Enfoque: El equipo se enfoca en la meta del Sprint
- Compromiso: Cada integrante se compromete a alcanzar sus objetivos en el Sprint
- Respeto: Los integrantes aceptan que los otros miembros son capaces e independientes.
- Accesible: El equipo es honesto frente al trabajo y los desafíos que se presenten

Los roles asignados durante el desarrollo son parte importante de Scrum y existen tres roles designados por los miembros del equipo: Product Owner, el responsable de maximizar el valor del producto final que genera el equipo de desarrollo. Development team, un equipo de profesionales que hacen el trabajo de entregar contenido potencialmente desplegable al final de cada sprint. Finalmente, el Scrum Master, es el encargado de promover y apoyar la metodología Scrum. Esto significa que se encarga de promover los valores, artefactos y prácticas.

Para los eventos, se tienen en cuenta los siguientes:

- Sprint: Un periodo de tiempo para el cual se propone un objetivo a finalizar.
- Sprint review: Al final del sprint, se realiza una reunión con todo el equipo donde se expone lo que este hecho y lo que no está hecho. En esta reunión se incluyen los clientes.
- Sprint retrospective: Una vez realizado el review, el equipo de desarrollo procede a una reunión donde cada integrante busca exponer tres puntos: Lo bueno realizado en el sprint, lo malo y como se puede mejorar.

Finalmente, Scrum presenta una serie de artefactos con la finalidad de mantener un orden y ver el progreso del proyecto. Entre estos encontramos:

- Historias de usuario: Definición principal de una funcionalidad a implementar.
- Tareas: Subdivisiones definidas para llevar acabo una historia de usuario.

- Sprints: Un lapso de tiempo definido para llevar acabo un número de tareas propuestas.
- Product backlog: Una lista donde se consignan las historias de usuario a realizar.
- Beta release: El objetivo o meta propuesta por el equipo de desarrollo.

Resonancia Magnética nuclear

La resonancia magnética nuclear, o NMR por sus siglas en inglés (Nuclear Magnetic Resonance) es un fenómeno que ocurre en el núcleo de átomos al ser expuestos a dos campos magnéticos, uno estático y el segundo oscilante. Este fenómeno, conocido como "Spin magnético" es una propiedad particular y suficientemente distinta de cada núcleo que se pueden diseñar experimentos de NMR para isótopos particulares de un elemento (Hornak, J. P., 1996-2017).

El spin magnético puede ser considerado como un pequeño campo magnético, el cual, al ser expuesto a los campos mencionados anteriormente mencionados, generará una señal de NMR. Es importante mencionar que no todos los núcleos producirán señales de NMR. Los núcleos cuyo número de protones y neutrones sean par, no tendrán reacción alguna en un espectro de NMR (Eldster. A.D., 2017).

Al inducir los campos magnéticos en el núcleo, el campo estático genera una polarización del spin en la molécula, mientras el oscilante, genera una transición en los spines de la molécula lo que los coloca en su estado de energía más alto posible. Al apagar el campo oscilante, ocurre el proceso de relajación de la molécula, volviendo a un estado de energía menor y generando una frecuencia resonante ligada a la transición del spin. Esta señal es medida por el reactor para ser señalada como un pico en un espectro de NMR (Nave, C.R., 2016).

Espectroscopía de resonancia magnética nuclear

Es una técnica utilizada para determinar los compuestos de una única estructura. Junto a esta técnica y otras como espectrometría de masas e infrarrojas, se puede obtener la estructura completa de una molécula.

Como vimos anteriormente, la molécula a estudiar, pasa por un proceso de excitación que aumenta su nivel de energía y un proceso de relajación con el cual vuelve a un estado de menor energía. El cambio de subida y bajada del spin en la molécula de

estudio es conocido como resonancia lo cual es la medida obtenida por el reactor. La gráfica generada por estas resonancias es conocida como un espectro (Reusch, W, 2013).

Existen datos analizables obtenidos de un espectro de NMR, entre los cuales tenemos:

- Número de señales: Número de átomos de un elemento en una molécula
- Posición de la señal: Cercanía a la molécula de muestra a comparar.
- Fuerza de la señal: Cantidad de concentración de cierto átomo en la molécula.

La espectroscopía de resonancia magnética nuclear puede identificar características especiales en moléculas, debido a esto, posee una amplia cantidad de usos como en el reconocimiento de origen de alimentos (Wist, 2015) o incluso en una clasificación por el lito tipo de rocas (Olatinsu, Olorode, Clennen, Esteban, Josh, 2017) puede ser logrado por medio de esta técnica.

Desplazamiento químico

Al pasar una molécula por NMR, actúa un campo magnético externo que altera el movimiento de los electrones en la molécula. Este movimiento, genera un campo magnético hacia el núcleo, en dirección contraria al campo magnético aplicado externamente. Este campo, crea un desplazamiento en la medida del espectro de NMR el cual depende exclusivamente del tipo de núcleo de la molécula y los electrones en movimiento en las partículas cercanas. Este desplazamiento, puede ser medido con la ayuda del NMR usando un elemento de referencia (IUPAC. 2016) y es conocido como Desplazamiento Químico (Chemical Shift) (Nave, C.R. 2016)

El cálculo del CS, puede ser usado para encontrar información relacionada con los enlaces químicos y la estructura de la molécula, por esto, se ha determinado como necesaria su inclusión dentro del proyecto.

Identificación de picos (Peak Picking) & Asignación automática

Debido al comportamiento gráfico de un espectro de NMR, es necesario identificar los valores máximos para obtener información acerca de la molécula, para esto, se usa una técnica conocida como identificación de picos la cual sigue una estrategia general que puede verse en (Castillo. A. 2015) cuya finalidad es la de extraer los parámetros del sistema Spin en la molécula.

Con la ayuda de la extracción de datos, existe la posibilidad de que un programa de computador pueda reconocer patrones de espectros para átomos, de este modo, se espera que, al asignar la evaluación de un espectro el programa pueda asignar los átomos correctos para cada sección del espectro en el que se encuentre un átomo o molécula.

Maven

La herramienta de construcción de proyectos open-source Maven, facilita el proceso de compilación y generación de ejecutables en proyectos. Sin embargo, Maven no se queda en el punto de compilar y generar un ejecutable para el proyecto. Con el uso de un fichero de configuración llamado POM (Project Object Model), el motor es capaz de construir correctamente el proyecto incluyendo las dependencias existentes con otros módulos adicionales usados por el proyecto. Debido a esto, los módulos auxiliares usados en el desarrollo también implementan Maven como su motor de compilación y ejecución. (Maven, 2002)

Maven posee un ciclo de fases por defecto al construir un proyecto:

1. **Validate:** Validación de la correctitud del proyecto y la información necesaria y disponible.
2. **Compile**: Compilación del código fuente
3. **Test**: Ejecución de las pruebas en el código fuente
4. **Package**: Empaquetar el código fuente en formato distribuible (JAR)
5. **Verify**: Ejecutar las pruebas necesarias para asegurar la calidad
6. **Install**: Instalación del paquete en el repositorio local para usarse como dependencia en otros proyectos.
7. **Deploy**: Envío al repositorio remoto para su distribución.

Sin embargo, el alcance del proyecto no cubre la distribución, por lo que se limita al ciclo de instalación con el comando:

- *mvn install*

Git Hub

Git Hub es una de las herramientas de control de versiones basado en git más usadas en el mundo. Esta provee una plataforma donde se aloja el código del proyecto *nemo2* y permite la posibilidad de creación de “ramas” que permiten a distintos

desarrolladores trabajar en copias locales del proyecto sin interferir o detener el progreso del otro. Gracias a esta plataforma, se logró el desarrollo en paralelo deseado por el equipo.

Adicionalmente, Git Hub es una herramienta de información para el equipo. Este permite la creación de “Issues” en los proyectos alojados. Los objetivos a desarrollar pueden ser creados como issues y a medida que se desarrollan, es posible realizar comentarios desde Git Hub. Al realizar un comentario, se obtenía un mensaje de correo automatizado que alertaba a los integrantes asignados en el issue acerca de un comentario en este. Git Hub permite crear “milestone” que corresponde a metas a alcanzar en los proyectos alojados, sea un 50% del proyecto, una mejora importante, una versión beta del producto.

Git es fácilmente instalado en ambientes de trabajo para lograr la “clonación” de proyectos alojados en la nube con el fin de obtener tener una copia local de estos. Al tener un repositorio local, el usuario es libre de realizar cambios, borrar, crear archivos del proyecto según su criterio sin afectar el repositorio remoto en Git Hub. El manejo de git es realizado principalmente por la consola de comandos o directamente desde el IDE preferido por los usuarios. Los comandos usados principalmente en un desarrollo con integración en git son:

- *git status*
- *git add [file_path]*
- *git commit -m "message of the changes sent to the remote repository"*
- *git push*

cada comando es precedido por la palabra “git”, esto indica que el comando siguiente pertenece a git. Status se encarga de listar todos los archivos para los cuales se han realizado cambios. Esto se debe a que muchas veces, no hay necesidad de enviar al repositorio cambios en archivos que pueden ser modificados por otros desarrolladores y generen conflictos, o simplemente hay archivos cuyas modificaciones no han sido terminadas por lo que no es recomendable enviar al repositorio remoto. Los comandos add, commit and push son parte de un ciclo de Git Hub.

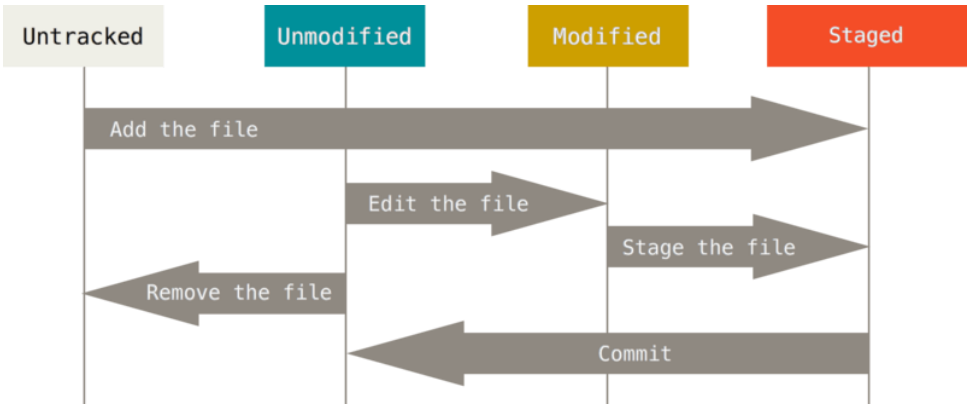

Ilustración 5 Estados para un archivo en git (tomado de: https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository)

Generalmente, los archivos en repositorios de Git Hub pueden tener dos estados: *En seguimiento* y *Sin seguimiento.* El último significa que Git Hub no tiene conocimiento de la existencia de este archivo. Usualmente, los archivos nuevos se encontrarán en este estado hasta que el desarrollador usa el add para informarle a Git Hub de la existencia del archivo y el deseo de su transferencia a estado en seguimiento. La razón de querer tener los archivos de un repositorio en estado *En seguimiento*, es el hecho de que git estará informado de todos los cambios realizados en el archivo que a su vez pueden ser consultados por el desarrollador. Un archivo que se encuentre en seguimiento por git, podrá tener tres estados. *Unmodified, Modified* y *Staged.* Los dos primeros estados se refieren a si un archivo tiene modificaciones o no. Git Hub realiza comparaciones de archivos frente al repositorio remoto del cual se origina. Si existen cambios, el uso del comando

- *git diff [file_path]*

Permite al desarrollador visualizar en la consola, una comparación entre las diferencias entre el archivo en el proyecto local y el alojado en el repositorio remoto.

Cuando el desarrollador esté seguro de que desea salvar los cambios en el repositorio remoto, se procede a realizar el comando add que no solo sirve para informar a git de realizar seguimiento de un archivo, sino que envía el archivo al estado *Staged* que significa la conformidad para ser enviado al repositorio remoto. Esto se logra con el uso del comando commit, que va acompañado de la partícula -m y un mensaje indicando el contenido de la actualización del repositorio remoto.

Una vez realizado estos dos comandos, se procede a realizar git push, que envía los cambios al repositorio remoto y realiza una sincronización del repositorio local con el commit que acaba de ser realizado. Cabe notar que aquellos archivos no incluidos en el commit persistirán en el repositorio local hasta que el desarrollador los requiera.

Sin embargo, es importante especificar qué tal como se observa en la **Figura 5**. es posible cambiar los estados de los archivos a gusto del desarrollador en caso de ocurrir un error al momento de usar el comando add.

Java

Java es un lenguaje de programación lanzado por Sun Microsystems en 1995, rápido, seguro, confiable y de distribución gratuita. Este es un lenguaje concurrente, basado en clases, orientado a objetos y diseñado para tener la menor cantidad de dependencias posibles.

Debido a la necesidad de actualizar un software creado en JavaScript, Java ha sido seleccionado para este proceso pues existen similitudes entre ambos lenguajes, con la finalidad de permitir a los desarrolladores reutilizar, en la medida de lo posible, el código generado en la primera versión de nemo.

Arquitectura

El esquema general de la arquitectura usada en la versión anterior de nemo y nemo2 se presenta en la siguiente figura:

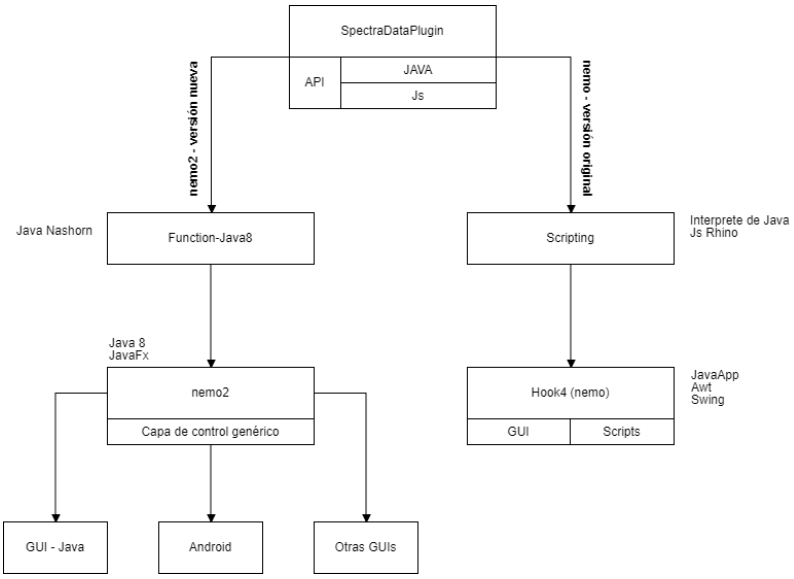

Ilustración 6 Esquema general de la arquitectura de nemo y nemo2.

Existe una librería principal SpectraDataPlugin que contiene las funciones de selección de picos, asignación automática y predicción de RMN existentes en nemo y nemo2. Estas funciones pasan por un intérprete de Java y Javascript en nemo2 y nemo respectivamente. En este punto, ya existe una diferencia, no solo es el uso de Java 8 para nemo2, Function-java8, su interprete, permite la extensión del código consignado en SpectraDataPlugin para ser usado en diferentes interfaces.

En este caso, nemo2 usa una interfaz desarrollada en Java 8 y JavaFX a comparación de nemo que usa Java applets, Awt y Swing que son tecnologías antiguas. Basado en el estudio realizado previamente al comienzo del proyecto nemo2, se decidió utilizar el tipo de arquitectura modelo-vista-controlador.

3.1 La arquitectura modelo-vista-controlador

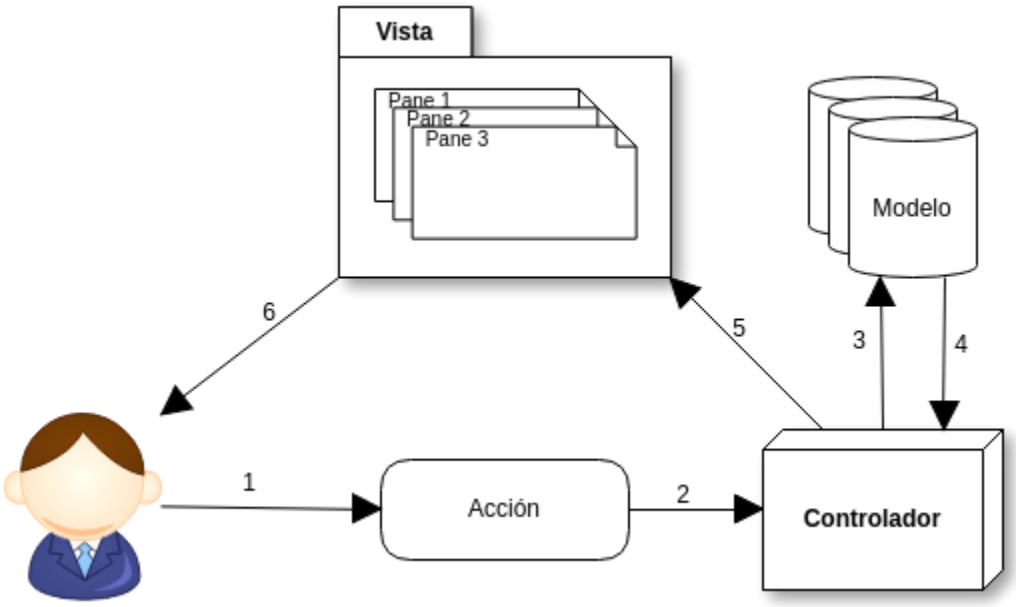

Ilustración 7 Esquema general de una arquitectura modelo-vista-controlador

Existe una diversa cantidad de arquitecturas para plantear el desarrollo de una aplicación. Entre las opciones disponibles, la idea de separar el procesamiento de datos y la parte gráfica de la aplicación, hace que nemo2 se incline favorablemente por el uso del modelo vista-controlador. Este modelo, como su nombre lo indica, divide una aplicación en tres partes. Una capa de modelo, que guarda toda la parte lógica de la aplicación (procesamiento, validación, asociación, procesamiento de los datos). La capa de vista, que realiza una presentación de los datos consignados en el modelo mientras se mantiene al margen de la capa de modelo. Finalmente, la capa de controladores, la cual gestiona las peticiones de un cliente y es responsable de dar respuesta con la ayuda de las capas de modelo y de vista.

Esta es la razón por la cual el equipo de desarrollo toma la decisión de centrar el desarrollo de *nemo2* a una arquitectura de tipo modelo-vista-controlador.

3.2 Modelo vista-controlador en nemo2

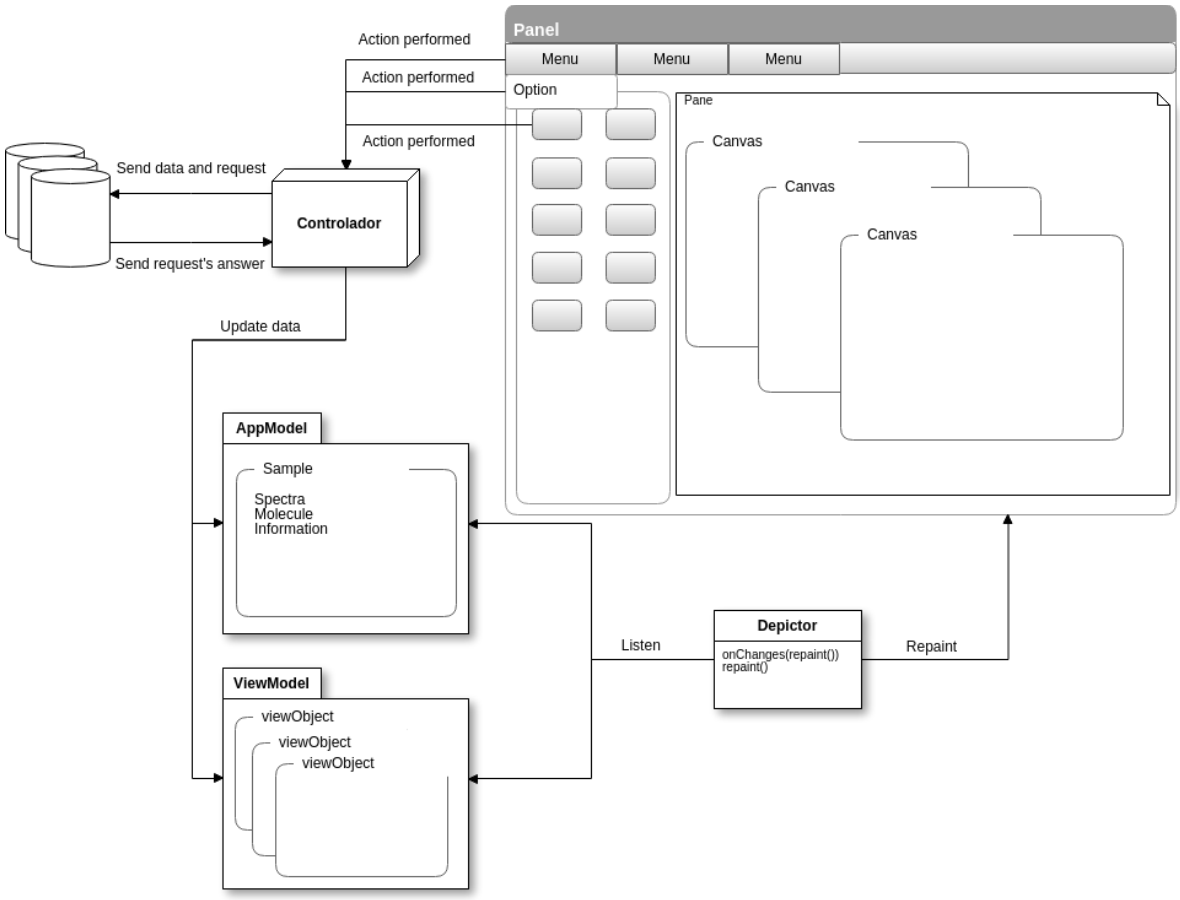

Ilustración 8 modelo-vista-controlador en el proyecto nemo2

La implementación del modelo sucedió de manera casi natural debido a la idea previamente deseada en la actualización de Hook4 que consiste en la separación de la parte lógica de la visual. A continuación, se presenta una descripción de cómo se implementa la arquitectura modelo-vista-controlador en nemo2.

Su interfaz en JavaFX es sencilla y separada por completo de los otros componentes de la aplicación. Esta se divide en tres partes, el área donde se muestra el espectro, sus cambios y sus demás características en la parte central. El área de menús, ubicada en la parte superior de la aplicación y finalmente, el área de acciones donde se consignan los íconos de cada función para la transformación del espectro o adición de información u objetos a este.

El controlador principal de la aplicación se encuentra en un archivo conocido como NemoController.java desde este, se inicia la aplicación, se carga la información de un espectro el cual se dibuja en su área correspondiente de la vista. Desde este controlador se crean los objetos que contendrán la información a evaluar o procesar en la aplicación (modelo). Cada acción a aplicar a un espectro tiene su archivo de instrucciones cuyo archivo es finalizado por el nombre de la función más “Action.java”. Estos archivos realizan un llamado a una función que envía la información del espectro activo junto con las acciones a realizar y una vez se obtienen resultados, se le ordena una actualización a la vista para la correcta visualización de los cambios realizados.

Generalmente, el procesamiento de datos se realiza gracias a un archivo especial “Sample.java” el cual es nombrado como la API de la aplicación nemo, el cual recibe una instrucción y la información del espectro activo para realizar llamados a las funciones consignadas en spectra-data-plugin el cual sirve como parte del modelo para la nemo2.

Adicionalmente nemo2 contiene una serie de objetos que estructuran el modelo de la aplicación. Existe un objeto principal, “ApplicationModel” que contiene las características actuales de la aplicación como alto, ancho y demás propiedades físicas de esta. Además, contiene un objeto importante llamado dataModel el cual almacena los datos del espectro activo. Esta clase realiza una instancia de la API Sample para obtener acceso a los métodos que realizarán transformaciones a la información. Al mismo nivel del dataModel, se encuentra otro objeto llamado viewModel, que a diferencia del dataModel, no guarda información alguna del espectro, pero guarda las propiedades estéticas o visuales que se presentan en la vista. La **Figura 8.** representa el esquema del uso de la arquitectura en nemo2.

2.3 Módulos involucrados

2.3.1 Exploración de Nemo (Hook4)

Debido a la naturaleza del proyecto, es necesario considerar la versión inicial de la aplicación a desarrollar: Hook4, mejor conocida como nemo. Como se ha mencionado anteriormente, esta fue realizada a base de Applets y el uso de JavaScript. El principal cambio a realizar es el uso de Java y la arquitectura general planteada para nemo2. Sin

embargo, es necesario enfatizar en que la gran mayoría de las funcionalidades planteadas en nemo2 están en nemo, por lo que es necesario dirigirse siempre a este módulo con la finalidad de identificar la forma de implementación, enfoque dirigido en el método y consideraciones necesarias para su funcionamiento.

Por lo anterior, el equipo determinó la necesidad de incluir una revisión a nemo para cada función a implementar. Si bien, las tecnologías cambian, nemo puede proveer una idea general para el desarrollo de las funciones en nemo2 o indicar qué funciones se deben usar en otros módulos para obtener los resultados deseados.

3.3.2 Intervención en spectra-data-plugin

Uno de los cambios principales en nemo2 es la implementación de un módulo aparte para el procesamiento de datos. Spectra-data-plugin es el módulo encargado de consignar las funciones que operan los datos y devuelven los resultados. Puede decirse, que el modelo planteado para nemo2, spectra-data es en efecto el modelo a usar.

Spectra-data es una recopilación de las funciones de procesamiento existentes en nemo más los cambios necesarios para permitir, con el uso de Maven, ser usada por otros módulos. Debido a esto, es de indispensable necesidad realizar búsquedas constantes para determinar qué archivos contienen las funciones necesarias para los procesos a implementar en nemo2.

Al ser la base principal de la transformación de datos para nemo2, este módulo no es definitivo. Durante el desarrollo de las funciones, pueden presentarse situaciones en las que es necesario modificar funciones existentes o crear archivos nuevos para métodos auxiliares, por lo que spectra-data es un módulo que tuvo cambios durante el transcurso de este proyecto. Esto mismo plantea la necesidad de estar constantemente explorando y/o modificando el módulo de spectra-data en cada una de las fases de exploración planteadas para los objetivos de implementación de funciones en nemo2.

2.3.2 Implementación de function-java-8

Un módulo desarrollado con la intención de realizar el scripting necesario para permitir la extensión de spectra-data-plugin a interfaces desarrolladas en una tecnología

diferente a la planteada en este documento. Es necesario incluirlo puesto que hace parte de las dependencias de *nemo2*, sin embargo, el proyecto actual no abarca un uso de este módulo, pero en el futuro lo hará.

3. Desarrollo de la herramienta

4.1 Actividades iniciales

Se inicia con una revisión al estado de la aplicación nemo2 en el momento de inclusión al proyecto. Esta posee funcionalidades básicas como la carga del espectro, carga de la molécula, opciones de zoom, ajuste del área visualizada, asignación manual de una molécula a un valor de integral, selección de un objeto dentro del área, expansión de un átomo para visualizar la cantidad de hidrógenos enlazados. Con base a esto, se comienza a definir los objetivos necesarios para la culminación de una herramienta funcional que se ajuste a los requerimientos del cliente.

Se definen tres funciones adicionales, que hacían parte de Hook4 como parte de la nueva versión de nemo: Realizar una función de peak picking, asignación automática y un predictor de desplazamientos. Para esto, se usará una API auxiliar que contiene los métodos necesarios para la ejecución de cada una de las funciones planteadas en los objetivos y al mismo tiempo separar la parte del procesamiento de datos de la interfaz.

4.2 Desarrollo

4.2.1 Ambiente de desarrollo

La metodología de desarrollo seleccionada por el equipo, incluyó características de prácticas ágiles (Específicamente Scrum) con la finalidad de hacer un desarrollo estructurado para llegar a la meta propuesta. Las características más importantes se describen a continuación:

- **Beta release**

Al tener en cuenta el tiempo de trabajo para el proyecto, se definió como meta final un Beta-release, el cual incluye las funciones descritas en este documento con su correcta implementación.

- **Product backlog**

Esta es la primera actividad a realizar en el inicio del primer semestre, donde se definen las historias de usuarios y sus objetivos a realizar al finalizar el tiempo de desarrollo de dos semestres.

- **Sprints**

Periodos de tiempo que abarcan un mes, para el cual se asigna una historia de usuario junto a sus tareas y se espera finalizarlas por completo o la mayoría. A lo largo del desarrollo de un sprint, se realizan revisiones del progreso junto al docente Andrés Castillo para identificar buenas prácticas o proponerlas en caso de poderse mejorar.

- **Reuniones retrospectivas**

Al finalizar cada sprint, se realiza una revisión final del avance en la historia de usuario. En caso de haber una implementación funcional, se realizaban ejecuciones de esta con el fin de observar el comportamiento, resultados y decidir el nivel de conformidad con el método finalizado o la proposición de modificaciones, mejoras o necesidad de reimplementación en caso dado que los resultados obtenidos no fueran semejantes a los esperados.

- **Historias de usuario**

Creada de acuerdo a las necesidades del cliente y del comportamiento esperado para una aplicación que realice transformaciones a espectros NMR. A cada historia de usuario se le asignó un valor basado en una sesión de planning-poker para dar un estimado del tiempo de trabajo requerido para su finalización.

- **Tareas**

Debido a que las historias de usuario abarcan un objetivo el cual requiere de un número de etapas para su correcta finalización, se crean puntos en específico a realizar para cada una, los cuales se conocen como tareas. Estas deben ser realizadas en un orden específico para el correcto desarrollo de la historia de usuario a la cual pertenecen.

- **Revisiones**

Al finalizar cada uno de los semestres, se realiza una ejecución de la aplicación integrando los nuevos cambios realizados. En esta ejecución, se busca identificar errores de compatibilidad o conflictos generados en métodos previamente funcionales y probados. Del mismo modo, se proponen soluciones en caso de existir y se procede a la creación de una nueva historia de usuario junto a sus respectivas tareas para llegar a una solución.

4.2.2 Equipo de desarrollo

El equipo de desarrollo está dividido en dos áreas. El equipo de programación, conformado por cuatro personas: Andrés Castillo, Roman Baer, Johan Benavides y Jhonatan Noguera. Los cuales poseen conocimiento en el desarrollo de software para llevar el desarrollo de la aplicación a su culminación. Un segundo equipo está conformado por estudiantes, docentes y profesionales del área de química.

4.2.3 Beta-release

Para el desarrollo de un objetivo final, se usó la herramienta web Git Hub, la cual permite la creación de una serie de puntos que pueden agruparse a una meta final, que para nuestro caso es el beta-release. De este modo, se realizó una reunión donde se especificaron los objetivos necesarios a cumplir para crear esta meta.

Aparte de la inclusión de las tres funciones descritas como objetivo en este proyecto, el beta-release de nemo2 incluye funciones fuera del alcance de este documento. Del mismo modo, la adición de características adicionales para obtener un índice de aceptación causa la posterior creación de nuevos casos de uso en el caso de que la funcionalidad presentará cambios significativos.

Adicionalmente a esto, se tuvo en cuenta las necesidades del cliente una vez se realizaron reuniones con ellos para definir objetivos de mayor prioridad o la adición de un nuevo objetivo que los clientes consideren esencial para la aplicación.

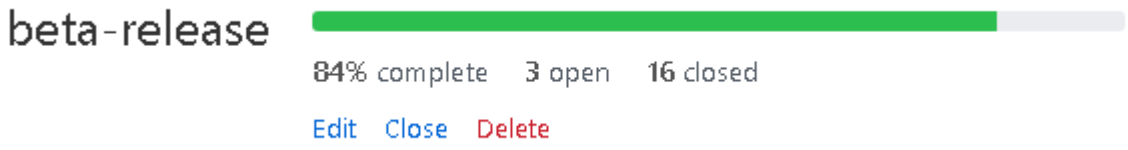

Ilustración 9 Milestone del Beta-release

4.2.4 Product Backlog

Una vez definido el Beta-release, se definieron como historias de usuario, los objetivos especificados en este documento gracias a la ayuda de la herramienta virtual Taiga.io. El equipo de desarrollo definió las tareas a realizar para cada historia de usuario y se determinó la prioridad de cada tarea e historia de usuario de acuerdo al impacto que tendrían sobre el beta-release.

Sprint	Tareas
Auto Peak Picking	Estudio previo
	Propuesta de solución
Auto Peak Picking 2	Desarrollo de la solución
	Implementación
NMR prediction	Estudio previo
	Propuesta de solución
NMR prediction 2	Desarrollo de la solución
NMR prediction 3	Desarrollo de la solución
	Implementación
NMR Assignment	Estudio previo
	Propuesta de solución
NMR Assignment 2	Desarrollo de la solución
	Implementación
Testing	Peak picking test
	NMR prediction test
	NMR auto-assignment test

Tabla 3 Sprints con las tareas a realizar del proyecto

La siguiente **Figura 10** representa un sprint en la herramienta Taiga.io en esta se creaban las tareas mencionadas anteriormente y de acuerdo al avance en estas, se ajustaba la tarea para llevar un orden en el desarrollo.

Ilustración 10 Detalle de un Sprint

El detalle de una historia de usuario se puede ver en la **Figura 11**. Aquí mismo se crean las tareas a realizar y se asignan a la persona encargada.

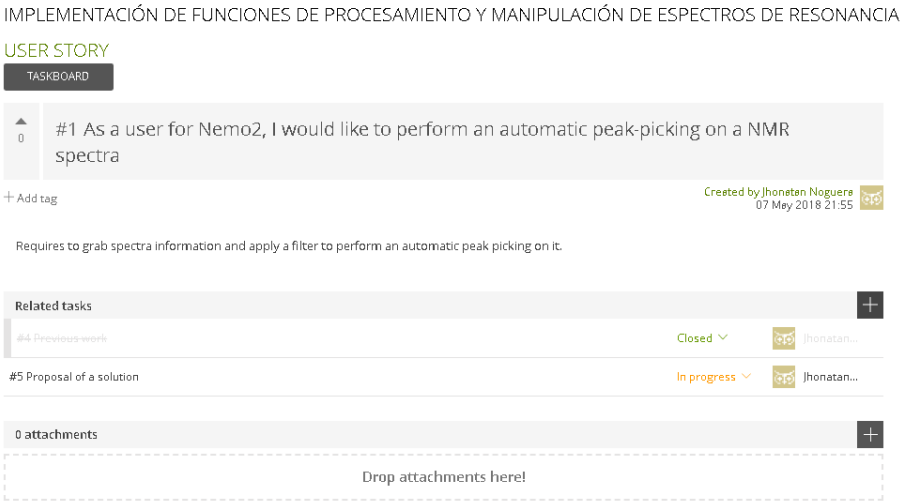

Ilustración 11 Detalle de una historia de usuario

Con el uso del backlog de Taiga.io, fue posible asignar historias de usuarios a los sprints necesarios, así como la asignación de las historias. La estimación, generada por una sesión de planning-poker, fue realizada al momento de definir las historias de usuario. Sin embargo, al avanzar en los sprints y hacerse más familiar con nemo2, Hook4 y el módulo de Spectra-data-plugin, la culminación de las tareas por cada historia de usuario fue más rápido y el tiempo pudo usarse en finalizar detalles de implementación y preparar con anterioridad el siguiente punto a realizar.

4.2.5. Procedimiento inicial

- **Maven**

Con la finalidad de realizar la compilación del código y lograr su ejecución vía IDE Eclipse (Eclipse, 2001). Existió la necesidad de agregar una condición en la ejecución del comando de instalación de Maven en los módulos adicionales para el proyecto.

- *mvn install -dskipTests*

Al correr el primer comando, Maven realizaba la mayor parte del ciclo de construcción hasta install. Esto significaba, que el proceso pasaba por las etapas de *Test* y *Verify* las cuales generaban errores debido a archivos de prueba existentes que intentaban probar métodos de versiones anteriores en esta librería por lo que se generaban conflictos y no era posible realizar la compilación o generación de archivos necesarios para ser usados por la aplicación *nemo2*. Este proceso fue necesario realizarlo cada ocasión que existieran cambios en los módulos de spectra-data-plugin o function-java8.

- **JavaFx & Java 8**

Hook4 fue desarrollada en Javascript, debido a esto, la decisión inicial en *nemo2* fue la de usar un lenguaje similar el cual no requiriera de cambios drásticos, por lo que la decisión del uso de Java 8 se hizo rápidamente. Sin embargo, el planteamiento de la arquitectura llevó a la elección de JavaFx para la implementación de la interfaz.

Si bien, el alcance del proyecto no abarca implementación con la interfaz desarrollada, se decidió añadirla como un trabajo adicional. Una vez seleccionadas estas tecnologías, fue requerido la instalación del Java Development kit (JDK), que, junto a Maven, permiten la correcta implementación de un ambiente de trabajo que puede ejecutar y realizar modificaciones a la aplicación *nemo2*.

- **Git Hub**

El proyecto nemo2, cuenta con un equipo de desarrollo de 4 personas. Debido a esto, existía la necesidad de poder trabajar en cambios de la aplicación sin afectar el rendimiento de otro integrante del equipo. Para esto, se decidió alojar el proyecto en Git Hub y facilitar el trabajo de independiente de los desarrolladores.

El uso de Git Hub en el proyecto puede ser considerado como altamente influyente. Para cada objetivo, se realizó una rama diferente en la cual se realizaba el desarrollo del objetivo pertinente y posteriormente se ejecutaba en las sesiones de revisión. De ser necesario realizar cambios o añadir nuevas características, el desarrollador continuaba en la rama del objetivo hasta cumplir con las expectativas. Una vez el equipo acordaba en la satisfacción con la solución del objetivo, se procedía a realizar un “merge” a la rama master. Esto se realizaba con la finalidad de evitar daños críticos en la última versión estable de la aplicación.

- **nemo2**

Al tener un ambiente de trabajo configurado para la edición y ejecución de *nemo2* y un lugar de alojamiento remoto para la aplicación, se procedió a clonar el repositorio donde se encuentra alojada la aplicación e igualmente se realizaron copias de los módulos spectra-data-plugin y function-java8. Esto fue posible gracias a git con el uso del comando

- *git clone [repo_url]*

Una vez clonados los repositorios, se procede a realizar la compilación y construcción de Maven en cada uno. Posteriormente, se importan los repositorios en eclipse con el uso de la opción de importar proyectos de tipo Maven. Una vez importados, se agregan en la propiedad Java build path de *nemo2* la referencia a los módulos adicionales. Finalmente, se realiza una prueba ejecutando la aplicación como una nemo application.

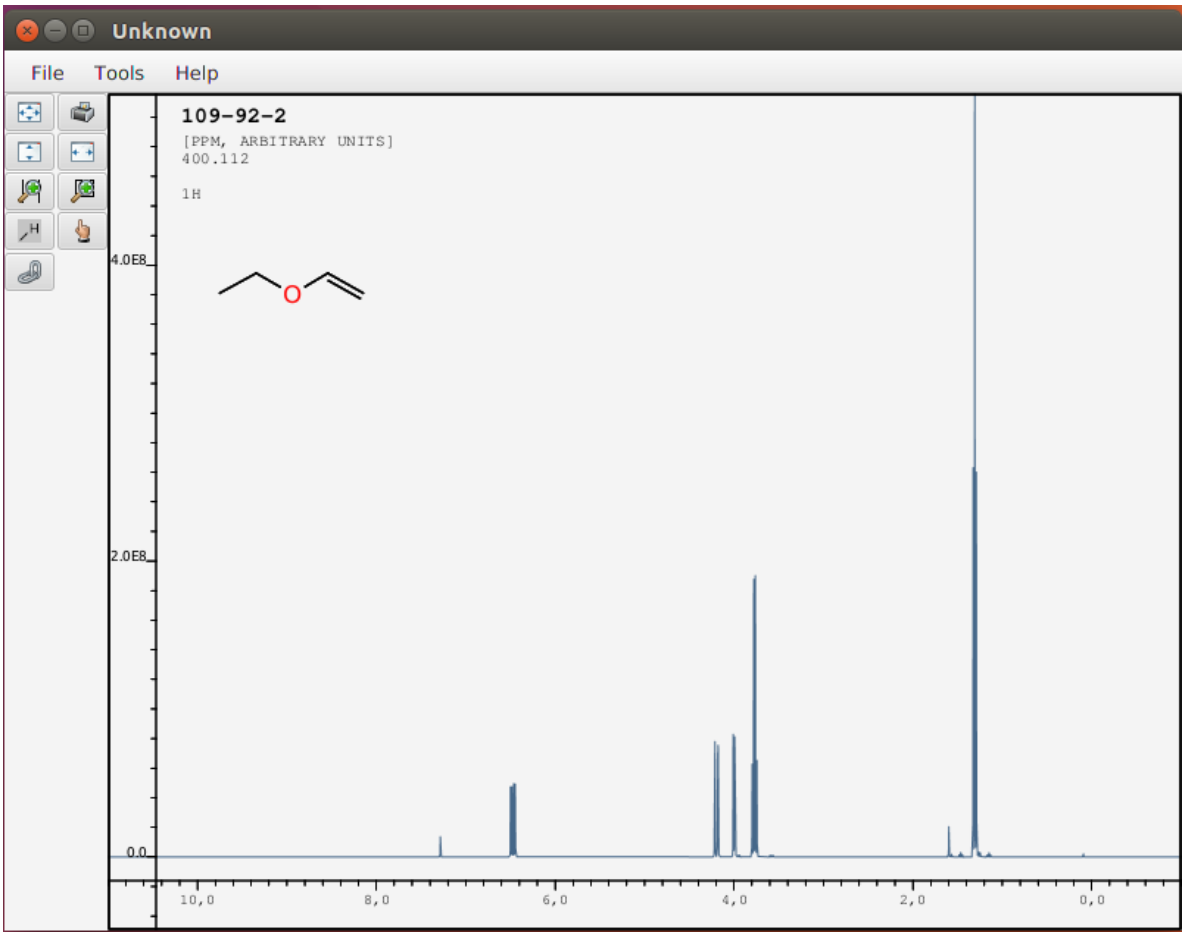

Ilustración 12 Estado inicial de la aplicación nemo2

La **Figura 12.** presenta el estado inicial de la aplicación una vez configurado el ambiente de trabajo y su ejecución vía Eclipse IDE. Este es el punto de partida para el desarrollo del proyecto descrito en este documento.

Para cada uno de los objetivos se definió una historia de usuario, las cuales se decidió dividir en 4 etapas para su culminación. El siguiente gráfico explica las etapas definidas:

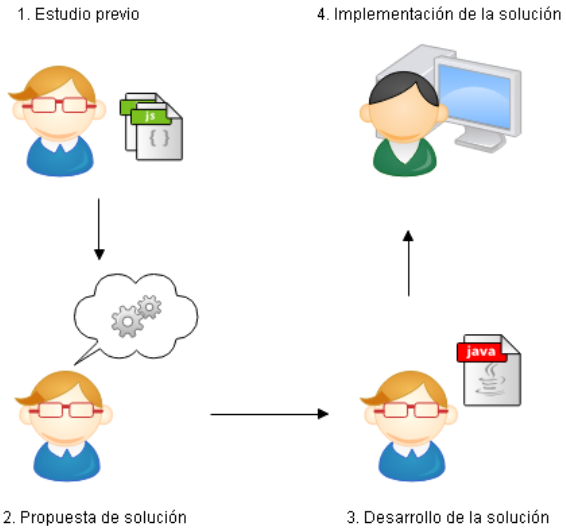

Ilustración 13 Esquema del proceso de solución de una historia de usuario

La primera fase se basa en el estudio del código realizado en Hook4 junto con las funciones existentes en el módulo de spectra-data-plugin con la finalidad de entender la forma de implementación (De existir) usada previamente. Esto se hace con el objetivo de formar una base para la segunda etapa que es una propuesta de solución.

nemo2 lleva una estructura previa la cual realiza llamados a un archivo API que se encarga de realizar las transformaciones al NMR basado en funciones del módulo spectra-data. Se busca seguir el mismo esquema de desarrollo, pero se permite, de ser necesario, la implementación de funciones adicionales para satisfacer el nivel de aceptabilidad decidido por el equipo de desarrollo.

Una vez definido un plan de solución, se avanza a la tercera etapa donde el desarrollador realiza el código necesario para resolver el problema propuesto en la historia de usuario. Finalmente, se realiza la cuarta etapa donde se implementa la función con la interfaz gráfica. Esto se decidió debido al hecho de que estas funciones realizan cambios visibles para el usuario que no son fácilmente identificables si no se pueden observar. Las fases tres y cuatros están ligadas muy fuertemente y no se cierran hasta que el equipo se encuentre a gusto con los resultados.

4.2.6. Desarrollo

Una vez definido el beta-release, el product-backlog y definidas las tareas a realizar por historia de usuario, se procede al desarrollo de las funciones especificadas como objetivos en el presente documento. En la siguiente tabla se presentan los resultados obtenidos:

Objetivo	Sección del resultado
Auto Peak Picking	4.2.6.1
NMR prediction	4.2.6.2
Auto Assignment	4.2.6.3
Unit tests	4.2.7.2
API	4.2.7.3

Tabla 4 Ubicación de los resultados en el documento

4.2.6.1 Peak picking

Estudio previo y propuesta de solución

La mayor parte de estas tareas se realiza en equipo. En la sección de módulos adicionales se explica el proceso realizado en la parte de estudio previo para los módulos de hook4 y spectra-data-plugin de la aplicación. Una vez finalizado el estudio, se plantea una propuesta de solución que para peak-picking se definió en las siguientes etapas:

1. Crear un botón de acción para el toolbar de la aplicación
2. Ligar el botón a un controlador que proveerá el contexto de la aplicación a un nuevo objeto
3. Realizar el llamado a la función deseada junto con los parámetros definidos anteriormente (null)

Una parte importante del desarrollo de *nemo2* es la idea de construir un archivo API que permita, en caso de mejoras o cambios drásticos en la aplicación, una fácil recuperación de las funcionalidades. Para esto, se adiciona un nuevo paso en la idea de implementación básica de la función.

- Realizar un llamado al archivo API el cual se encargará de realizar la ejecución de las funciones en spectra-data-plugin.

Un paso adicional que definitivamente ayudará a los trabajos futuros a realizar en *nemo2.*

Desarrollo de la solución

Según lo planteado en la sección anterior, se debe iniciar por la implementación del botón en la interfaz de la aplicación. Para esto, se agrega un nuevo campo en el archivo NemoToolbar.fxml. Este archivo es base para la realización de la interfaz en JavaFX. Aquí se consignan la distribución y localización de un panel que contiene las funciones de la aplicación nemo2. Aquí se registra un nuevo botón por medio de lenguaje HTML que contiene nombre, ícono, localización, entre otras características necesarias para su correcta visualización en el panel.

Una vez modificado el archivo, se puede realizar una ejecución a la aplicación para ver que se halla incluido un nuevo botón en el área designada para las funciones.

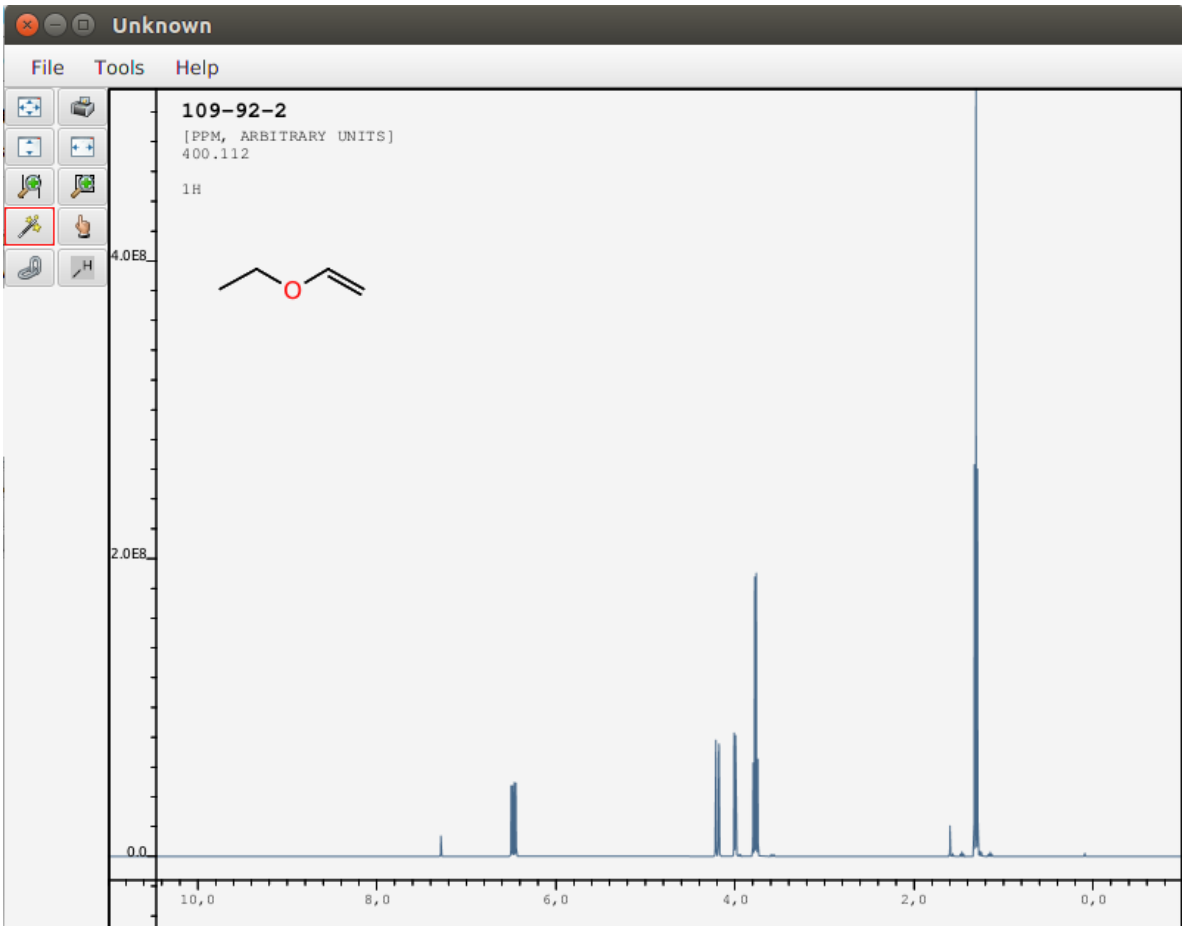

Ilustración 14 Implementación del botón para la función de peak-picking

Sin embargo, en el estado actual, el botón no es funcional por lo que se requiere proceder a la definición del controlador que accionará el evento de presionar el botón. Una vez más, el archivo ya para realizar esta definición existe en la aplicación, por lo que se debe modificar el archivo NemoToolbar.java en el proyecto creando un nuevo botón usando la etiqueta "@FXML" lo cual indica a la aplicación que se debe ligar a un botón con el mismo nombre definido en un archivo de tipo FXML el cual se definió anteriormente. Posteriormente, se procede a incluir en el método init de NemoToolbar.Java la instrucción a realizar en caso de existir un evento sobre el botón creado.

Para esto, se decidió la creación de archivos aparte cuyo nombre es escrito como *Action.Java, siendo * el nombre de la función a realizar, los cuales contendrán la información necesaria para realizar la acción que se espera. Debido a esto, se procede a la creación de un archivo de nombre: AutoPeakPickingAction.Java. Al igual que las funciones existentes en la aplicación, se debe proveer un contexto a la función, este

contexto es en efecto el ApplicationModel, previamente mencionado, que contiene la información esencial de que se está visualizando en nemo2 y sus caracteristicas.

Esto se realiza definiendo la creación de un nuevo objeto de tipo AutoPeakPickingAction como la acción a realizar una vez ocurre un evento en el botón creado previamente. Como parámetro a este nuevo objeto, se le pasará al ApplicationModel, lo que define un constructor para AutoPeakPickingAction que reciba un parámetro de tipo applicationModel y pueda cargar los datos de este correctamente. Una vez cargados los datos correctamente, procedemos a ejecutar la función deseada. Para esto, necesitamos usar la información del espectro guardada dentro del dataModel que se encuentra en el applicationModel. Gracias a la arquitectura de la aplicación, es posible obtener la información de esta, gracias a una instancia de Sample.Java, se tiene acceso a los métodos consignados en este archivo, sin embargo, es desde AutoPeakPickingAction que se llama la función dentro de Sample.java junto con el parámetro deseado, el cual es nulo en el caso a implementar.

Inicialmente, este acercamiento tomado para la función fue correcto. La función desarrollada realizaba correctamente las evaluaciones deseadas en los espectros de prueba usados (Etil vinil éter y Etil benceno). El equipo de desarrollo aprobó la función y los cambios realizados se mezclaron junto a la versión a enseñar en los demos. Sin embargo, al comenzar el desarrollo de la función de auto asignación, peak-picking sufrió ligeros cambios para realizar los procesos basados en un valor por defecto (100) o el número de hidrógenos en caso de existir una molécula activa que representa el espectro activo.

Implementación de la solución

Gracias a los pasos definidos en la propuesta de solución, la implementación se realiza en los primeros pasos mientras se define el botón en el archivo FXML y se crea el controlador. Una vez finalizado el proceso de desarrollo de la función, se procede a la ejecución del código. Es importante mencionar que la aplicación debe tener activa un espectro 1D, ya que la función utilizada no permite la determinación en espectros 2D. Igualmente, la necesidad de enviar el contexto de la aplicación para determinar los valores deseados, no permite el uso de la función en el caso de que no exista un espectro activo.

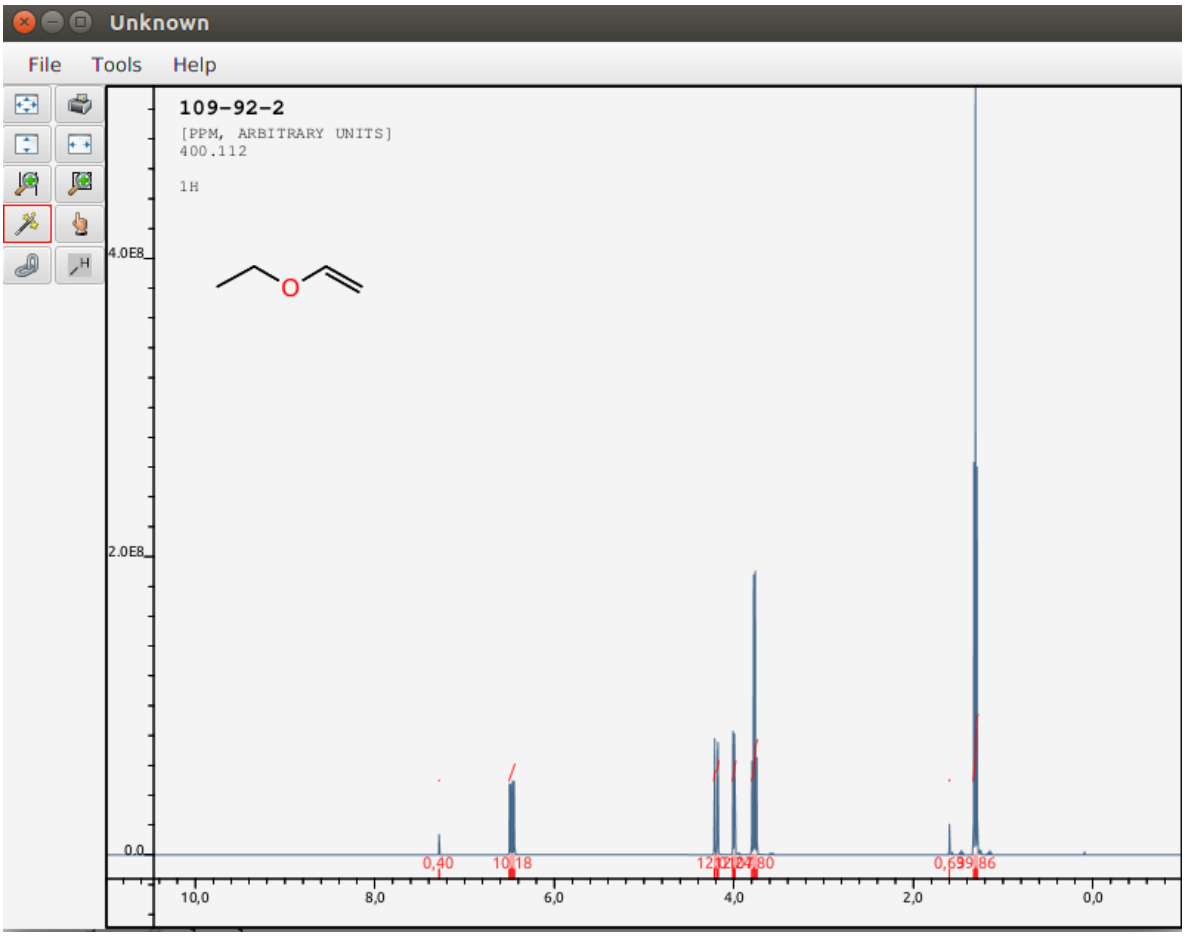

Ilustración 15 Resultados de usar la función peak-picking en un espectro de Etil vinil eter

4.2.6.2 NMR prediction

Estudio previo y propuesta de solución

La mayor parte de estas tareas se realiza en equipo. En la sección de módulos adicionales se explica el proceso realizado en la parte de estudio previo para los módulos de hook4 y spectra-data-plugin de la aplicación.

Terminada la exploración en el módulo de funciones adicionales y la versión previa de la aplicación, se usa la información adquirida para terminar de definir los detalles de desarrollo. En este caso, encontramos que el archivo SD.Java en spectra-data-plugin permite ser instanciado fácilmente. Segundo, debido a que la acción de predicción de un NMR se realiza basado en un archivo, se decide que no es recomendable la inclusión de un botón en la barra de acciones y se acuerda agregar la opción en uno de los menús desplegables de la aplicación.

Ahora, se debe plantear la solución a usar, de tal modo que sea posible seguir una implementación similar a funciones existentes en la aplicación. Para esto, se decidieron los siguientes pasos para la implementación:

1. Crear la opción en uno de los menús desplegables de la aplicación
2. Ligar el botón a un controlador que proveerá el archivo de molécula para realizar la predicción de esta
3. Realizar un llamado al archivo API el cual se encargará de realizar la ejecución de las funciones en spectra-data-plugin.

Desarrollo de la solución

Según lo planteado en la sección anterior, se debe implementando el botón en uno de los menús desplegables en la aplicación. Para esto, se agrega un nuevo campo en el archivo Nemo.fxml. Este archivo es base para la realización de los menús en la interfaz desarrollada en JavaFX. Aquí se consignan la distribución y localización de los menús de opciones a los que se tiene acceso al ejecutar *nemo2*. Aquí se registra una nueva sección por medio de lenguaje HTML que contiene nombre, ícono, localización, entre otras características necesarias para su correcta visualización en el menú.

Una vez modificado el archivo, se puede realizar una ejecución a la aplicación para ver que se haya incluido una sección para la acción a realizar en el menú “Tools” de la aplicación.

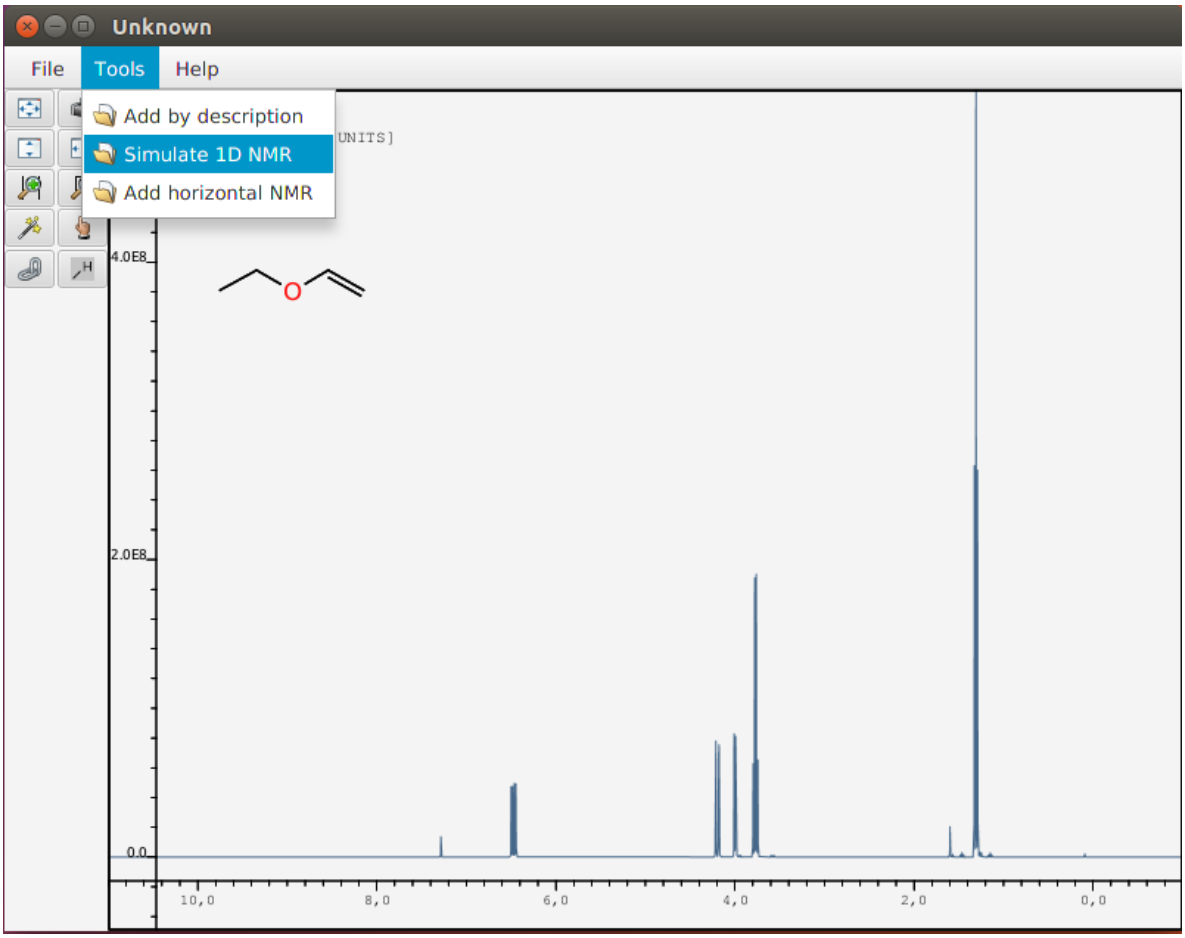

Ilustración 16 Implementación de una sección en el menú para la función NMR prediction

Sin embargo, al igual que lo sucedido en la funcionalidad anterior, esto no es suficiente para que la aplicación realice el procedimiento deseado. Se determinó que no había necesidad de crear un controlador adicional o un archivo separado para culminar los procedimientos necesarios. Por el contrario, se usa el archivo NemoController ya existente que incluye los controladores de las opciones alojadas en los tres menús disponibles de la aplicación.

La función a implementar debe ejecutarse mediante un evento, para esto, en el archivo Nemo.Fxml se usa el nombre de la función como el objeto a realizar en el momento de una acción. Posteriormente, la predicción se desarrollará basada en una molécula que el usuario debe proveer. Este paso implica que se debe implementar una forma de permitir al usuario navegar entre sus archivos para seleccionar la molécula para la cual desea se

realice la predicción de su NMR. Sin embargo, esto se desarrolla fácilmente usando características ya incluidas en Java.

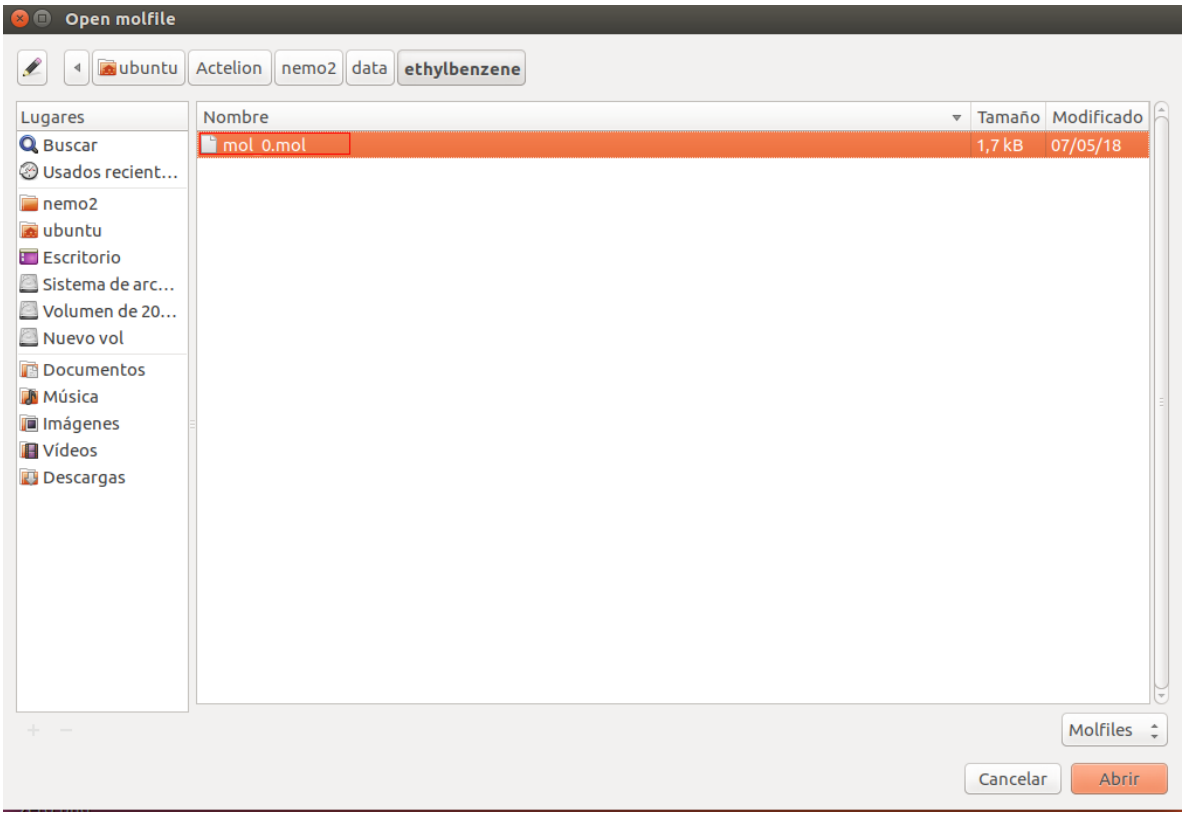

Ilustración 17 Ventana emergente que permite la selección del archivo .mol para la función NMR prediction

Como se observa en la figura 17, el archivo a cargar, debe ser un objeto de extensión *.mol que contendrá información desde la cual, nemo2 podrá añadirla dentro de applicationModel para su posterior manipulación.

Posteriormente, se pasa a crear un objeto de tipo Sample.java el cual permite cargar la información de la molécula que al mismo tiempo es guardada dentro del contexto actual de la aplicación. Se genera la molécula sin expansiones de hidrógenos y se procede a hacer uso de una nueva instancia de SD.Java para crear una predicción del espectro NMR de la molécula proveída por el usuario. Se decide posteriormente realizar un proceso de peak picking a la molécula usando un proceso similar al realizado para la función ya implementada junto al código necesario para que el canvas principal de *nemo2* realice los cambios necesarios para la visualización en este.

Una vez terminado el desarrollo de la función, el equipo decidió añadir al espectro generado un peak-picking junto a una asignación automática. Esto se da, debido a que los datos del NMR fueron generados por la propia aplicación, Esto significa que se identifica fácilmente que átomo de la molécula genera un pico en el espectro. En base a este razonamiento, la asignación de átomos al espectro es realizada sin mucho percance usando la información consignada en el objeto NMRSignal1D dentro de la instancia de Sample.Java previamente creada. Al haber realizado previamente un peak-picking, no se tiene percance alguno en añadir el código necesario para añadir esta información en la instancia de Sample activa. De igual modo, se realiza la adición de la función principal de predicción, sin la asignación o peak-picking, al archivo API Sample.Java

Implementación

Al igual que en la función anterior. Gran parte de la implementación se adquiere durante el proceso de desarrollo debido a la importancia de visualizar las transformaciones en la interfaz para comprobar el funcionamiento del método.

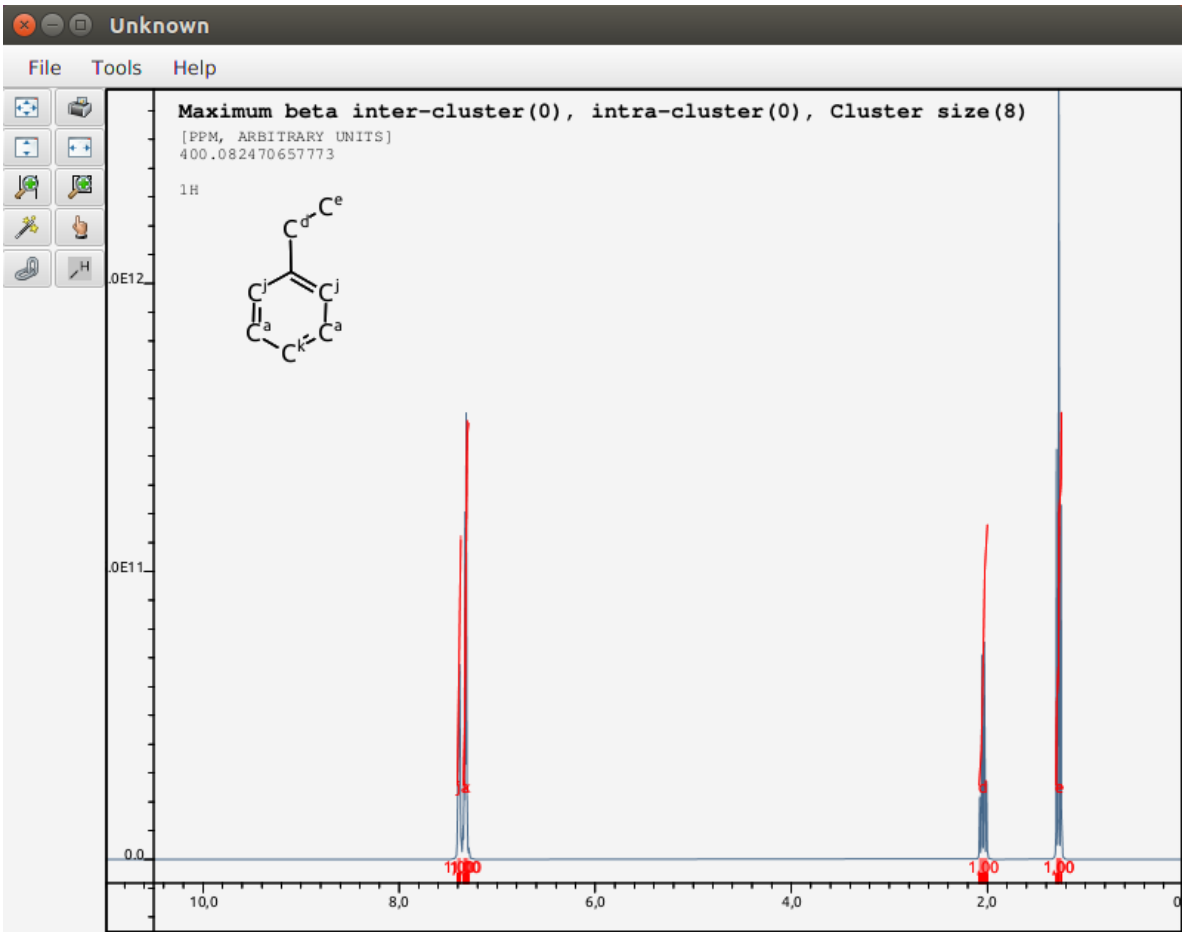

Ilustración 18 Ejecución de la función NMR prediction basado en una molécula de Etil benzeno.

4.2.6.3 NMR Auto-assignment

Estudio previo y propuesta de solución

La mayor parte de estas tareas se realiza en equipo. En la sección de módulos adicionales se explica el proceso realizado en la parte de estudio previo para los módulos de hook4 y spectra-data-plugin de la aplicación.

Terminada la exploración en el módulo de funciones adicionales y la versión previa de la aplicación, se usa la información adquirida para terminar de definir los detalles de desarrollo. En este caso, encontramos que nuevamente,el archivo SD.Java en spectra-data-plugin contiene extensiones de los paquetes de predicción que serán útiles en el desarrollo de la auto asignación. Por otro lado, la asignación se realiza en base a un espectro NMR activo y a una molécula que describo dicho espectro, lo que lleva a concluir que la inclusión de un botón en la barra de acciones es la forma apropiada de proveer al cliente una forma de realizar la acción de auto asignación.

Ahora, se debe plantear la solución a usar, de tal modo que sea posible seguir una implementación similar a funciones existentes en la aplicación. Para esto, se decidieron los siguientes pasos para la implementación:

1. Crear un botón de acción para el toolbar de la aplicación
2. Ligar el botón a un controlador que proveerá el contexto de la aplicación a un nuevo objeto
3. Realizar el llamado a la función deseada junto con los parámetros necesarios para su funcionamiento
4. Llamado a la función desde el Archivo API Sample.Java

Desarrollo de la solución

Una vez más, se debe iniciar por la implementación del botón en la interfaz de la aplicación. Se agrega un nuevo campo en el archivo NemoToolbar.fxml que permita la visualización de un nuevo botón por medio de lenguaje HTML en el panel donde se encuentran los demás botones de funciones para un espectro. Una vez modificado el archivo, se ejecuta la aplicación para observar si la posición es la deseada, el ícono, tooltip entre otras características sean las deseadas.

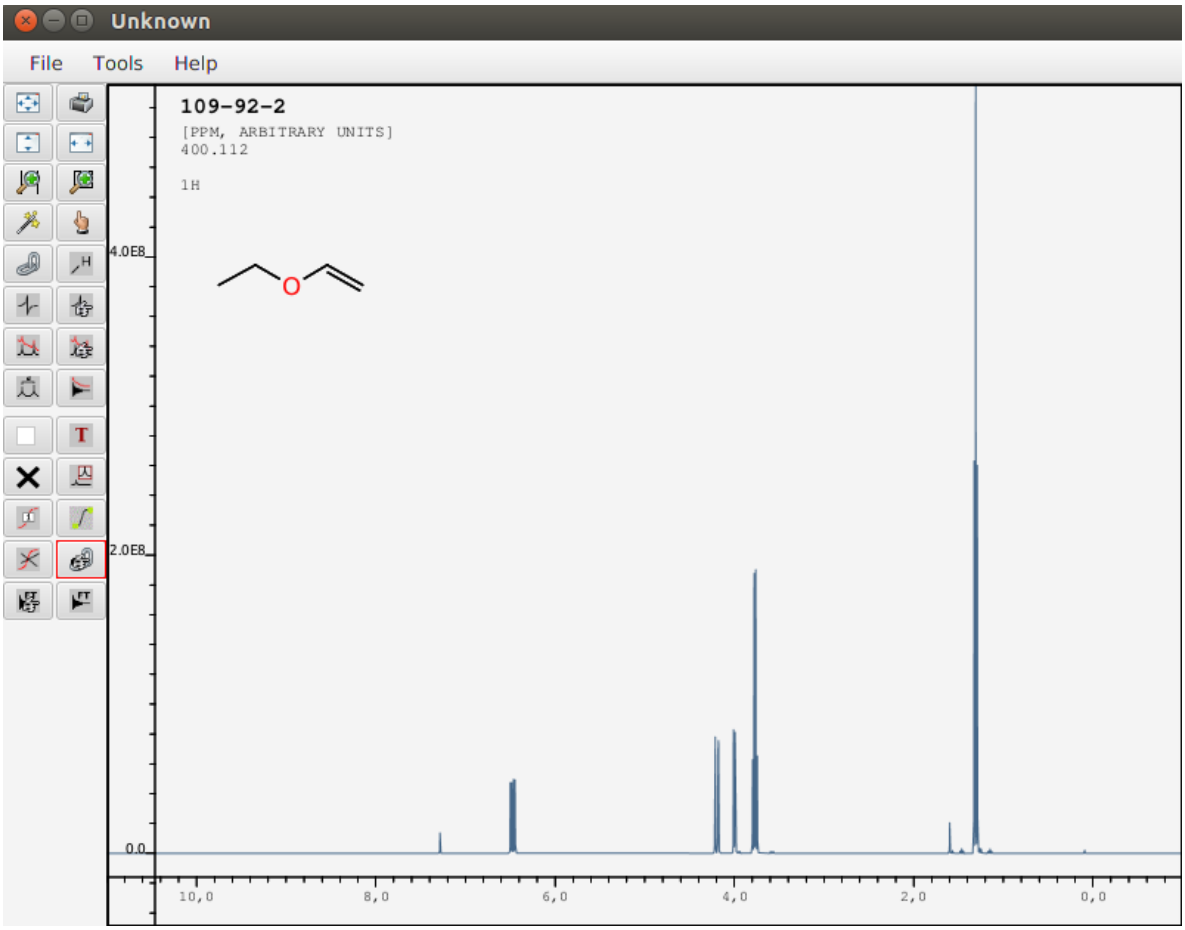

Ilustración 19 Implementación del botón para la ejecución de la función NMR auto-assignment.

Una vez el nivel de conformidad del equipo con el botón creado es suficientemente satisfactorio, se procede a definir el botón dentro del archivo NemoToolbar.java junto a la inclusión en el método init de este, que permitirá la comunicación con el archivo donde se encuentra la acción a realizar en el método de auto asignación.

Al terminar las modificaciones necesarias para la visualización del botón en la interfaz junto a la creación de su controlador, se procede a la creación del archivo AutoAssignAction.Java que será el encargado de tomar la información provista desde el controlador para ejecutar las funciones usando una instancia de la API Sample.Java.

Para esta función, se requiere que exista un peak picking realizado en la información del sample para realizar la correcta asignación. Esto llevó a realizar ligeras modificaciones en el método de Auto peak-picking, que consistió en normalizar los resultados de las

integrales, basado en el número de hidrógenos de la molécula que representa el espectro. Se debe notar que, de no existir una molécula activa, el auto peak-picking sigue realizándose en base a 100 por defecto.

La auto-asignación también requiere tener una molécula presente, por lo que se extrae la información de esta desde el applicationModel que se envía en el controlador. Del mismo modo, se extrae la información del peak-picking realizado y por medio de Sample.java, se ejecuta la función que realiza una predicción de señales a usar al momento de hacer la asignación que una vez más se encuentra en el archivo API Sample.Java la cual usa dos funciones auxiliares para la construcción del objeto a evaluar escritas en AutoAssignAction.Java.

Implementación

La implementación de esta función se realiza mientras ocurre el proceso de desarrollo, puesto que la forma de verificar si se realizaba una asignación o no, era desde la interfaz, marcando cada señal en el espectro con una letra que estaba igualmente asignada a uno de los átomos de la molécula marcando la referencia a esta.

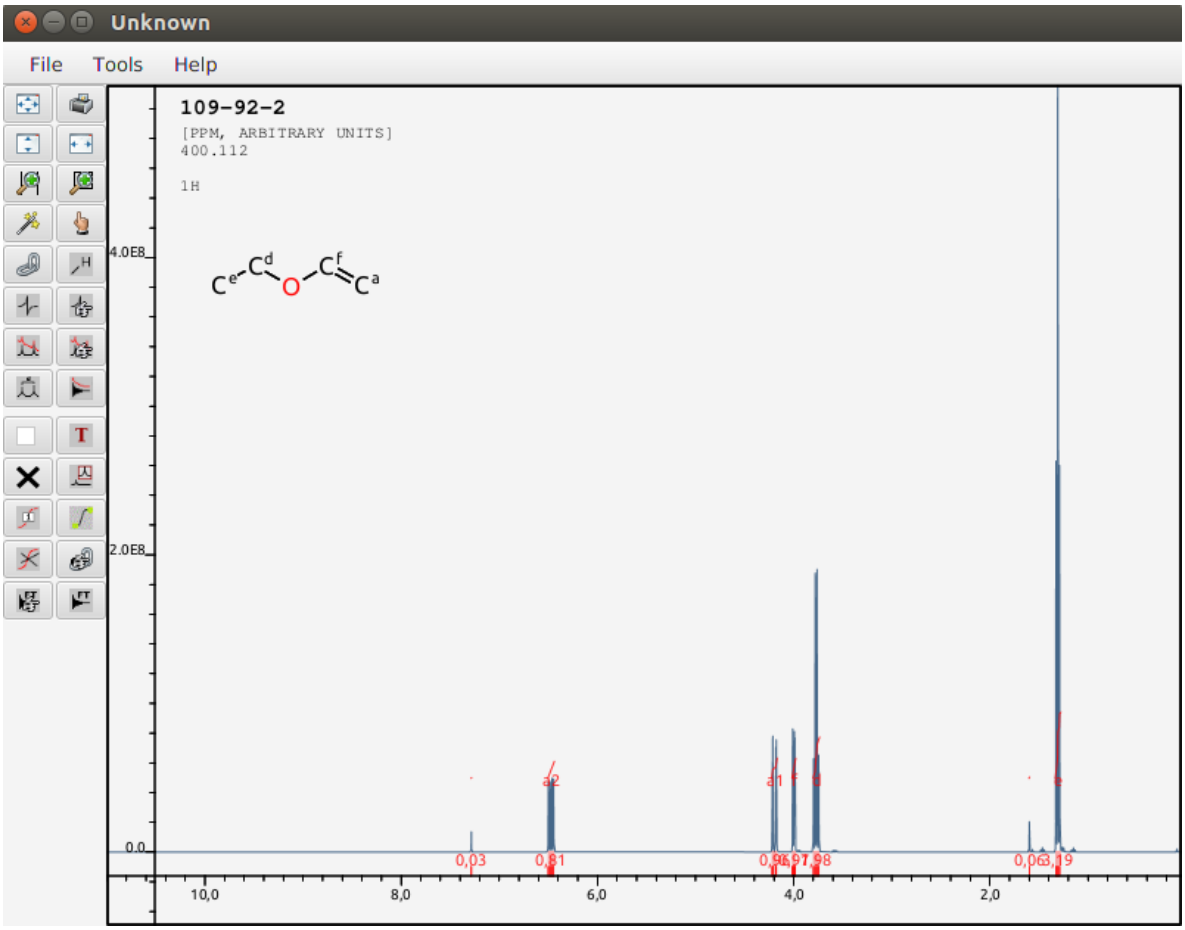

Ilustración 20 Ejecución de la función NMR auto-assignment a un espectro y molécula de Etil vinil éter.

4.2.7 Conclusión del desarrollo

4.2.7.1 Manual de usuario

Nemo2 necesita ser instalada en un ambiente local y es dirigida a un público cuya finalidad sea el análisis y manipulación de espectros de resonancia magnética nuclear. Una vez correctamente instalada, el usuario se enfrentará a una interfaz como la visualizada en la **Figura 20**

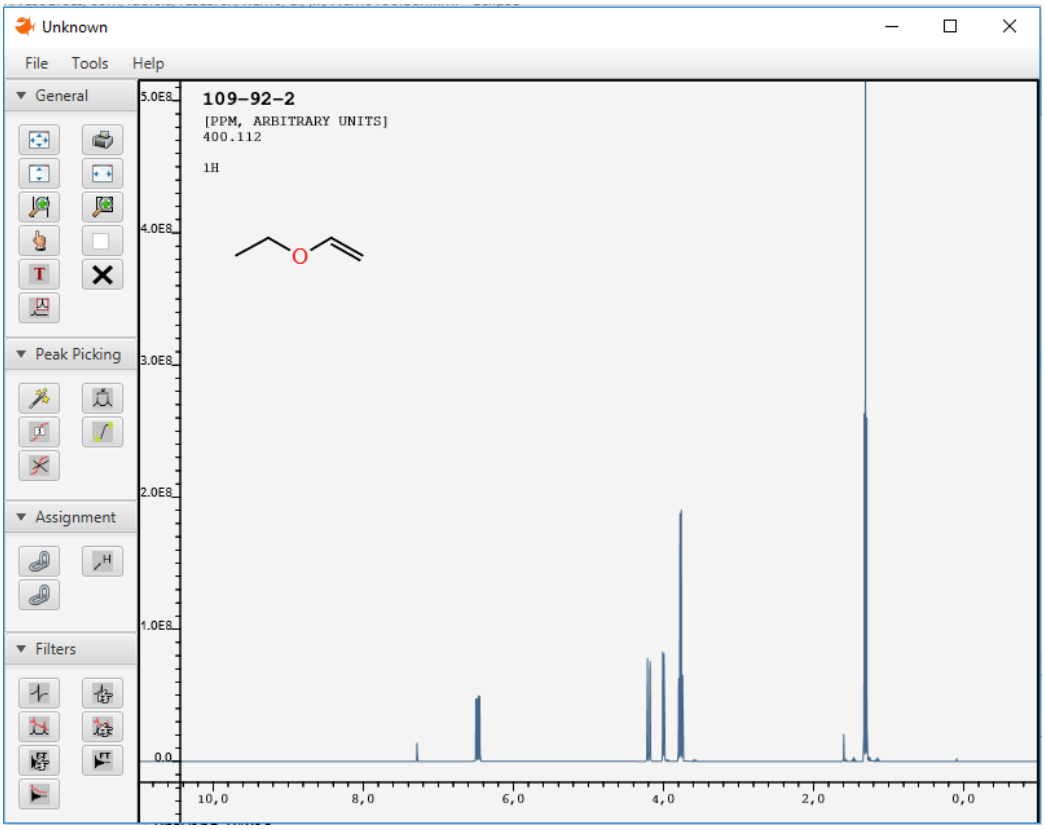

Ilustración 21 Interfaz final del proyecto Nemo2

La parte superior de esta ventana contiene la barra de menús, desde la cual se puede acceder a funcionalidades como carga o salvado de espectros, predicción de NMR, posibilidad a exportar en SVG entre otros. Al lado izquierdo, se observan botones agrupados bajo unas categorías decididas por el equipo de desarrollo. Finalmente, al lado derecho, de mayor tamaño que las áreas descritas anteriormente, se encuentra el canvas principal. Este es compuesto de un grupo de canvas para su manipulación de manera individual.

En la sección de menús, “Help” permite al usuario acceder a una documentación más detallada de las funciones implementadas, su forma de ser ejecutadas, información de desarrolladores entre otros.

4.2.7.2 Pruebas

Una parte importante es la de probar la ejecución de las funciones implementadas. Para esto, se decidió usar la herramienta Junit para generar pruebas de unidad. Una ventaja

adicional de Maven, es que no hay necesidad de instalar un ambiente para trabajar con Junit, funciona solo con agregar la dependencia al archivo POM creado por Maven.

Sin embargo, al momento de realizar las pruebas, se encontró una dificultad. Debido a la complejidad de los objetos y la cantidad de cifras en los datos que se guardan en la aplicación, realizar pruebas a los datos no fue viable. Por lo que las pruebas fueron enfocadas en identificar que los resultados obtenidos al realizar una transformación fueran guardados correctamente dentro del contexto de la aplicación, por lo que, en cada prueba realizada, se buscó verificar que los datos existieran, ya que estos no existen antes de realizar la acción. Los resultados de las pruebas se pueden ver en la **Figura 22.**

```
Results :

Tests run: 7, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 10.695 s
[INFO] Finished at: 2018-05-20T16:46:32-05:00
[INFO] Final Memory: 10M/188M
[INFO] ------------------------------------------------------------------------
```

Ilustración 22 Ejecución de las pruebas unitarias

4.2.7.3 API

Previamente se mencionó el deseo de crear una API desde la cual se pudieran instanciar los métodos que se representan de manera gráfica en la interfaz de nemo2. La idea planteada con esto está vinculada al uso del módulo function-java-8 el cual permite usar diferentes tecnologías para desarrollar una nueva interfaz que pueda mostrar la representación de los datos consignados en el modelo. La implementación de este archivo API tiene como objetivo facilitar la idea de la posibilidad de usar tecnologías diferentes a JavaFX. Si las funciones se realizan mediante paso de información y la instanciación de un objeto, no se ve la necesidad de modificar los métodos existentes para una actualización futura como se vio necesario en las funciones desarrolladas en el presente documento.

La implementación de dichas funciones fue realizada de manera sencilla, no se encontraron contratiempos y solo se vió la necesidad de hacer que el controlador usará las funciones consignadas en el archivo API para determinar que en efecto no existía problema alguno al usar dicho acercamiento.

4.2.7.4 Trabajos futuros

Como se ha mencionado anteriormente, existe la idea de extender la aplicación nemo2 para permitir la incorporación con interfaces realizadas con tecnologías diferentes, entre las cuales se consideran swing y android. Esto con la finalidad de permitir a los usuarios de nemo2 tener acceso a la aplicación en formas más versátiles.

Hook4 permitía la integración de funciones con espectros 2D lo cual no está contemplado en el marco del desarrollo de nemo2. Como adición de los trabajos futuros, se planea implementar las funcionalidades provistas en hook4 para espectros 2d en su nueva versión, nemo2.

A pesar de cumplir con el objetivo, realizar un testeo más profundo sobre las funcionalidades de nemo2 es una de las consideraciones a tomar para un futuro. No solo su correcto funcionamiento sino la facilidad con que un usuario pueda interactuar con la herramienta se contempla entre los trabajos a realizar.

5. Conclusiones

La finalidad que abarca el documento no es solo la de realizar una actualización a la herramienta hook4. Al desarrollarse este proyecto bajo el planteamiento de un trabajo de grado para pregrado, se tienen las siguientes conclusiones:

Las ciencias de la computación abarcan un amplio espectro en las áreas de trabajo encontradas comúnmente. Previo a la integración al desarrollo de este proyecto, no existía una idea clara de cómo se puede integrar una ciencia exacta como la química al área de computación. Una vez culminado el desarrollo planteado para este proyecto, se tiene un concepto más sólido de la fácil integración entre otras áreas de investigación y las ciencias de la computación.

Si bien las metodologías de desarrollo tradicionales aún están presentes, la gran mayoría de empresas involucradas en el desarrollo de software implementan metodologías ágiles gracias a los beneficios que estas traen. Para el desarrollo de este proyecto, se usaron características de la metodología Scrum. Previamente en cursos de desarrollo de software se expuso la forma de trabajo bajo esta metodología ágil, sin embargo, la forma de ejecución fue casi mínima con respecto a los artefactos como reuniones, revisiones entre otros. Al poner en práctica esas características, se evidencia personalmente los beneficios

que traen estas durante el desarrollo de un proyecto por lo que la opinión con respecto a trabajar bajo estos lineamientos es considerada como experiencia valiosa para el desarrollador.

Debido a la naturaleza del proyecto, este no solo es una experiencia gratificante de aprendizaje para el estudiante. El sentido de responsabilidad y confianza otorgada al estudiante para realizar avances en el desarrollo de manera remota marcan una clara línea entre la academia y la vida laboral que se considera importante para la transición entre estas.

Posteriormente, el trabajo en equipo fue inculcado en el desarrollo del estudiante debido a la naturaleza del campo de acción. El trabajo en grupo para el desarrollo de este proyecto se ve beneficiado gracias a esto, es importante señalar como una de las habilidades inculcadas en el transcurso de la academia, la lengua inglesa, beneficia el intercambio de información con miembros del grupo que no son hispanohablantes nativos.

Perseverar hace parte del desarrollo del trabajo realizado en este documento. Se evidencia que no es fácil el comprender código cuando no se tienen conocimientos previos de las tecnologías a usar y en especial cuando es un código adquirido por parte de un tercero. Sin embargo, perseverar en adquirir los conocimientos necesarios para proseguir con los objetivos es gratificante cuando se es capaz de llegar a una solución satisfactoria.

6. Bibliografía

1. Chang, R. , 2013, *Chemistry*, 11th ed. New York: McGraw Hill,, ISBN 978-0-07-340268-0 pp. 2
2. A. Castillo, 2015, "Robust Automatic Assignment of Nuclear Magnetic Resonance Spectra for Small Molecules," Ph.D. dissertation, Dept. Ing. Sist. e Ind., Univ. Nacional de Colombia , Bogota DC, Colombia.
3. Castillo, A.M., Bernal, A, Dieden, R., Patiny, L. y Wist, J., 2016, "Ask Ernö", *J. Cheminform.,* DOI: 10.1186/s13321-016-0134-6.
4. ACD/Labs. (n.d.). Visitado en Septiembre 13, 2017, de http://www.acdlabs.com/
5. Mestrelab Research S.L. - Analytical Chemistry Software Solutions. (n.d.). Visitado en Septiembre 13, 2017, de http://mestrelab.com/
6. One Moon Scientific. (2016). *nmrViewJ*. Visitado en Septiembre 13, 2017, de: http://www.onemoonscientific.com/nmrviewj
7. Solutions, P. (n.d.). PERCH Solutions. Visitada en Septiembre 13, 2017, from http://new.perchsolutions.com/
8. Hornak, J. P. (1996-2017). *The basics of NMR.* Visitado en Septiembre 13, 2017, de: https://www.cis.rit.edu/htbooks/nmr/inside.htm
9. Hunt, D.I. (n.d.). *Nuclear Magnetic Resonance (NMR) Spectroscopy.* Visitado en Septiembre 13, 2017, de: http://www.chem.ucalgary.ca/courses/350/Carey5th/Ch13/ch13-nmr-1.html
10. Eldster. A.D. (2017). *Predicting Nuclear Spin (I).* Visitado en Septiembre 13, 2017, de: http://mri-q.com/predict-nuclear-spin-i.html
11. Nave, C.R. (2016). *HyperPhysics.* Visitado en Septiembre 13, 2017, de: http://hyperphysics.phy-astr.gsu.edu/hbase/Nuclear/nmr.html
12. The Apache Software Foundation. Visitado en Mayo 21, 2017, de https://maven.apache.org/what-is-maven.html
13. Reusch, W. (2013). *Nuclear Magnetic Resonance Spectroscopy.* Visitado en Septiembre 13, 2017, de: https://www2.chemistry.msu.edu/faculty/reusch/virttxtjml/spectrpy/nmr/nmr1.htm
14. *Proton Nuclear Magnetic Resonance Spectroscopy.* (n.d.). Visitado en Septiembre 13, 2017, de: http://web.chem.ucla.edu/~harding/notes/notes_14C_nmr02.pdf
15. Arana, V. A., Medina, J., Alarcon, R., Moreno, E., Heintz, L., Schäfer, H. y Wist, J., 2015, "Coffee's country of origin determined by NMR: The Colombian case", *Science Direct. Food Chemistry,* vol 175, pp. 500-506.
16. Olatinsu, O.B., Olorode, D.O., Clennel, B, Esteban, L. y Josh, M., 2017, "Lithotype characterizations by Nuclear Magnetic Resonance (NMR): A case study on limestone and associated rocks from the easter Dahomey Basin, Nigeria", *Science Direct. Journal of African Earth Sciences,* vol 129, pp. 701-712.
17. Harris, R. K., Becker, E. D., M., C. D., Granger, P., Hoffman, R. E., & Zilm, K. W. (1970, January 01). Further conventions for NMR shielding and chemical shifts (IUPAC Recommendations 2008). Visitado en Septiembre 13, 2017, de: https://www.iupac.org/publications/pac/80/1/0059/
18. Nave, C.R. (2016). *HyperPhysics.* Visitado en Septiembre 13, 2017, de: http://hyperphysics.phy-astr.gsu.edu/hbase/Nuclear/nmrcsh.html
19. *Migrating from Java Applotes to plugin-free Java technologies*. (January, 2017). Visitado en Septiembre 13, 2017 de: http://www.oracle.com/technetwork/java/javase/migratingfromapplets-2872444.pdf
20. C. (2017, febrero 19). Cheminfo/hook4. Visitado en septiembre 13, 2017, de https://github.com/Cheminfo/hook4
21. Cadavid Andres, Martinez J. D., Velez Jonathan, (2013), Revisión de metodologías ágiles para el desarrollo de software. Visitado en septiembre 13, 2017, de: https://ojs.uac.edu.co/index.php/prospectiva/article/viewFile/36/21
22. Eclipse Foundation. Visitado en Mayo 21, 2017, de http://www.eclipse.org/